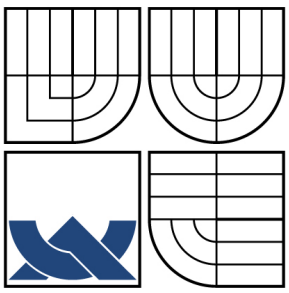
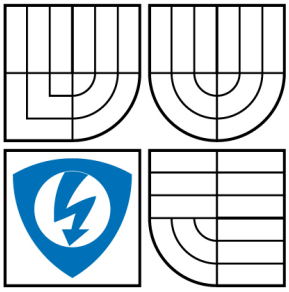


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

MODERNÍ KONCEPCE SYSTÉMU VIRTUÁLNÍ TŘÍDY

MODERN CONCEPTION OF VIRTUAL CLASS SYSTEM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

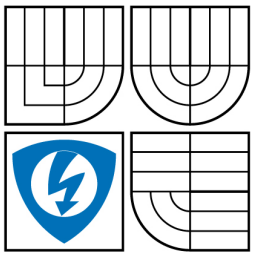
AUTOR PRÁCE — Bc. PAVEL OUTĚŘICKÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. VÁCLAV PFEIFER

SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Pavel Outěřický	***ID:***	89158
Ročník:	2	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Moderní koncepce systému virtuální třídy

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku virtuální třídy a popiště možnosti současných systémů. Uveďte jaký přínos má virtuální třída a jaké jsou aplikační možnosti a praktické využití v praxi. Na základě teoretických znalosti systému virtuální třídy rozeberte možnosti síťové komunikace mezi klienty v rámci virtuální třídy. Analyzujte jaké jsou síťové nároky na plynulou funkci virtuální třídy v případě připojení velkého množství uživatelů. Nastudujte technologii multicastu a broadcastu a diskutujte jejich praktickou aplikaci v rámci systému virtuální třídy. Dále rozeberte problematiku videokonference a její praktické nasazení v rámci VT. Na základě teoretických znalostí navrhněte koncept vlastního systému VT a za použití programovacího jazyku JAVA koncept realizujte. Vámi vytvořený systém VT aplikujte v rámci libovolného víceúčelového serveru a proveďte jednoduchá měření, která budou prezentovat vlastnosti VT.

DOPORUČENÁ LITERATURA:

[1] Spell, B.: JAVA Programujeme profesionálně. Nakladatelství CPRESS 2002. ISBN: 80-7226-667-5.
[2] Bigelow, S., J.: Mistrovství v počítačových sítích. Nakladatelství CPRESS 2004. ISBN: 80-251-0178-9.

Termín zadání:	9.2.2009	***Termín odevzdání:***	26.5.2009

Vedoucí práce: Ing. Václav Pfeifer

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Práce se zabývá problematikou distanční interaktivní výuky s využitím systému virtuální třídy, jakožto softwarového komunikačního nástroje. Po krátkém rozboru základních typů mezilidské komunikace a výuky je zde úvodem popsána virtuální třída z úplně základního pohledu ve smyslu prvotního vysvětlení co tedy virtuální třída je, k čemu slouží a jaké přínosy může přinést. Níže jsou pak rozebrány možnosti, podporované nástroje a funkce dnešních komerčních i bezplatně dostupných systémů tohoto typu s názornými ukázkami a se zaměřením na možnosti síťové komunikace. Další částí dokumentu je popis a rozebrání implementačních možností pro tvorbu virtuální třídy, jako čistě webového systému nebo jako desktopové aplikace se zaměřením na možnosti využití různých programovacích jazyků či prostředí. Dále je jako často používaná funkce u virtuální třídy také popsána videokonference a nároky na její implementaci.

Druhá část, která tvoří asi tři čtvrtiny dokumentu, je věnována popisu mé vlastní praktické realizace systému virtuální třídy, jakožto desktopové aplikace komunikující přes serverovou aplikaci s využitím jazyka Java. Je zde na úvod popsán základní návrh tohoto systému s jeho implementovanými nástroji pro vzdálenou výuku. Níže jsou pak v kapitolách postupně všechny tyto implementované nástroje a funkce podrobně popsány, jak z pohledu potenciálního uživatele, tak i z pohledu softwarového vývojáře.

Klíčová slova:

výuka, komunikace, interakce, software, Internet, Java, klient, server

Abstract

This thesis is focused on problems of an interactive distant education with usage of virtual classroom as a software communication tool. The short analysis of types of human communication and education is followed by a description of virtual classroom systems in a very basic point of view like what the virtual classroom is and what its usage is and what advantages these systems have. Furthermore, potentialities and supported tools and functions of today's commercial and also non-commercial products are described including illustrative examples and focus to potential types of network communication implementation. Next section of the thesis presents possibilities of virtual classroom software development as completely web-based application or as desktop application where usage of different programming languages is considered. After that, videoconference as a often used tool in virtual classroom is described and its demands on the ways and types of realization and implementation are stated.

The second part of this thesis creates about three quarters of whole document and is dedicated to description of my own realization of virtual classroom system as a built-in-java desktop application communicating across a built-in-java server desktop application. The first section of this part introduces a basic design and concept of the system including its implemented tools for the distant education. The following chapters describe in details each implemented tool and function from the potential user point of view and also developer point of view.

Key words:

education, communication, interaction, software, Internet, Java, client, server

Bibliografická citace této práce

OUTĚŘICKÝ, P. *Moderní koncepce systému virtuální třídy.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 77 s. Vedoucí diplomové práce Ing. Václav Pfeifer.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

V Brně dne …….. ….…….
podpis

Poděkování

Na úvod bych rád poděkoval především svému vedoucímu Ing. Václavu Pfeiferovi za cenné připomínky a rady, které mě vedly k úspěšnému dokončení této práce. Mé díky patří také kamarádům Ing. Tomáši Hulovi a Ing. Janu Vilímkovi za několik užitečných rad, které vyplynuly z příležitostných konzultací během našich setkání. Dále pak bych rád poděkoval mé partnerce MUDr. Janě Míčkové, která mi mnohokrát obětavě pomáhala při testování v průběhu vývoje mého systému. A v neposlední řadě také všem ostatním přátelům a kolegům, kteří mi byli nápomocni při testování připojení více uživatelů.

Obsah

Seznam obrázků

1 Virtuální Třída

Pro mnohé může pojem virtuální třída okamžitě znamenat přepnutí toku myšlenek směrem do tématiky objektově orientovaného programování. Anglický překlad tohoto pojmu, jak je použit zde, dodá přesnější znění a význam, o jaký se bude jednat v této práci – Virtual Classroom. Nejde tedy o virtuální třídu ve smyslu objektu v nějakém programovacím jazyku, ale, jak naznačuje ono „room“, o třídu ve smyslu místnosti plné studentů, kteří poslouchají svého přednášejícího a mají možnost s ním komunikovat, pokládat dotazy a sdělovat své myšlenky. Slovo „Virtual“ pak naznačuje, že nejde o obyčejnou „kamennou“, zdmi a okny tvořenou třídu, ale o třídu, která hmotně neexistuje – Virtuální třídu.

Taková třída je tedy jakási „místnost“ ve virtuálním prostoru Internetu, kde se mohou účastníci „sejít“ a komunikovat, stejně jako by byli spolu v obyčejné místnosti, například s lavicemi, tabulí a oním pedagogem přednášejícím látku. Je realizována softwarem jako interaktivním nástrojem, který umožňuje lidem se tímto způsobem setkávat a komunikovat.

K Virtuální třídě (dále jen VT) se může účastník připojit svým způsobem odkudkoliv. Je k tomu zapotřebí mít jen patřičné vybavení, které v tomto případě zahrnuje počítač připojený dostatečnou rychlostí k Internetu a vybavený potřebným prohlížečem, případně sofistikovaným softwarem, dále dle potřeby sluchátka s mikrofonem a u VT podporujících videokonferenci také webovou kameru.

Pro přihlášení zadá účastník do webového prohlížeče či do klientské aplikace adresu, na které je VT provozována a přihlásí se na svůj účet do patřičné studijní skupiny. Výběr studijní skupiny může být realizován formou seznamu s předmětem a jménem vyučujícího. Po přihlášení už může plně využívat podporovaných funkcionalit. Ve VT se používá pestrá škála možností pro interaktivní výuku, mezi něž patří například „whiteboard“, což je jakási plocha sdílená všemi uživateli, kterou může přednášející používat k okamžitému grafickému doplnění látky, zkrátka to, co je klasická tabule či projekční plátno v „kamenné“ třídě. Dále pak možnosti například sdílení dokumentů, vlastních prezentací, video ukázek, možnost pokládání písemných i ústních dotazů, chatování mezi jednotlivci či skupinami, zobrazování doplňkových informací atd. Toto byl jen částečný výčet toho, co všechno může VT umět. Každý systém VT má však své vlastní možnosti a funkcionality. Ale všechny VT mají jednu společnou důležitou vlastnost. Umožňují totiž kolaborativní způsob výuky s okamžitou odezvou.

1.1 Stimulace, asynchronní a synchronní výuka

Při komunikaci a tedy i při vzdělávání rozlišujeme dva druhy komunikace. Asynchronní a synchronní. Asynchronní komunikace probíhá tak, že účastníci spolu nekomunikují okamžitě a tzv. „naživo“. Asynchronní komunikace tedy spočívá v tom, že se předá nějaká zpráva a příjemce si ji později vyzvedne. Mezi tuto komunikaci se řadí například psaní e-mailů, různá diskusní fóra na webových stránkách, nástěnka atp. Stejně tak je tomu i v e-learningnových systémech, kdy určité webové stránky slouží pro ukládání studijních materiálů, které si pak studenti mohou stáhnout a nastudovat. Tento typ výuky a tedy i komunikace má výhodu v tom, že tyto informace jsou zpětně použitelné, jsou dobré pro opakování a samostudium. Ovšem samostudium či opakování z daných skript postrádá jistou stimulaci.

Na druhou stranu máme komunikaci synchronní (nebo též online), kdy uživatelé spolu komunikují „naživo“ ve stejném čase a dochází tak tedy k přímé interakci. Tímto typem komunikace je například telefonní hovor, VoIP, okamžité posílání zpráv, tedy chat atp. A právě synchronní výuka dodává jistou stimulaci, protože student je nucen být na výuce v danou dobu přítomen a také okamžitě reagovat na případný dotaz. Stejně tak jako jsou studenti v běžných učebnách ve škole. No a tímto typem výuky, tedy synchronní, se právě zabývají systémy virtuálních tříd.

1.2 Softwarová řešení VT

Dnešní doba skýtá pro vývoj systému typu VT mnoho možností. Vývojáři nejsou omezeni pouze na jeden typ či na jeden jazyk. Vzhledem k tomu, že systémy VT v sobě mohou kombinovat poměrně pestrou škálu nástrojů a funkcí, tak také jejich architektura má mnoho podob. Dle mého názoru se tak nějak nedá říci, že tento či onen typ systému VT je nejpoužívanější či nejrozšířenější. Jediná, co asi lze tak nějak s jistotou říci je, že všechny typy těchto systému, ať už při vývoji či při používání, mají své klady a své zápory.

1.2.1 Webový prohlížeč či desktopová aplikace?

Základní rozdělení typů bych udělal asi v tom, jestli klient na svém počítači používá pro připojení a práci ve VT pouze webový prohlížeč nebo má nainstalovaný sofistikovaný program v podobě klientské aplikace. Spousta VT je vytvořena právě jako webová služba, pro kterou k připojení stačí mít pouze onen webový prohlížeč. Takovéto typy jsou tvořeny většinou s využitím PHP nebo také pomocí Java servletů a apletů. Používá se kombinace Java servletů či apletů s klasickými html stránkami, tedy dohromady JSP (Java Server Pages). Webové servery, na kterých tyto aplikace běží, jsou většinou propojeny s nějakým typem databázového serveru, kde uchovávají data. Systémy VT, které z klientského pohledu používají pouze webový prohlížeč, mají tu výhodu, že není nějak zapotřebí vyvíjet přímo klientskou aplikaci, kterou by si museli všichni klienti stáhnout či jinak získat. Ovšem na druhou stranu velkou nevýhodou těchto systému je možná nekompatibilita prohlížečů či nutnost na klientském počítači stejně doinstalovat například nějaký plug-in pro správnou funkci. Grafické uživatelské rozhraní VT často nebývá zrovna nejjednodušší a mnohdy bývá poměrně komplikované (myšleno z pohledu vývoje, ne z pohledu koncového uživatele) a tedy jeho implementace do prohlížeče (který svým základním principem slouží k prohlížení hypertextových stránek), tak aby bylo kompatibilní s různými typy prohlížečů nemusí být úplně jednoduchá.

Na druhou stranu je zde možnost vytvoření vlastní klientské desktopové aplikace, kde vytvoření grafického rozhraní je jasně dané danou aplikací a pokud se aplikace na počítači spustí, tak bude vždy stejná. Nastává samozřejmě problém s přenositelností na různé platformy. Z tohoto důvodu je například vhodné právě využití programovacího jazyka Java či .NET. Pokud je k systému vytvořena také nějaká vlastní serverová aplikace, tak zde bych se přikláněl spíše k vytvoření pomocí C++, protože u serverové aplikace nejde zas tak o multiplatformitu, ale spíše o rychlost V čem oproti Javě a .NETu rozhodně vedou aplikace psané přímo v C++.

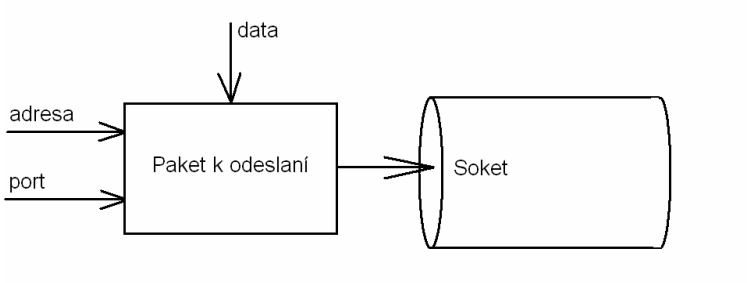
1.2.2 UDP a TCP komunikace

Vzhledem k tomu, že hlavním smyslem VT je komunikace a tedy komunikace přes IP sítě, uvádím zde malý náhled UDP a TCP komunikace. Jak bude uvedeno později, tak mnou vytvořená VT je realizována pomocí jazyku Java a proto uvádím také implementaci UDP komunikace v Javě.

V síťové komunikaci rozlišujeme dva základní typy přenosu dle toho, jaký transportní protokol používají. Jsou to TCP (Transmission Control Protocol) nebo UDP (User Datagram Protocol). Protokol TCP garantuje spolehlivé spojení přes vytvořený kanál, který je vytvořen po celou dobu spojení. Naproti tomu při použití UDP není zajištěno, že pakety budou doručeny nebo že dojdou ve správném pořadí. O to se tedy musí případně postarat aplikace na aplikační vrstvě.

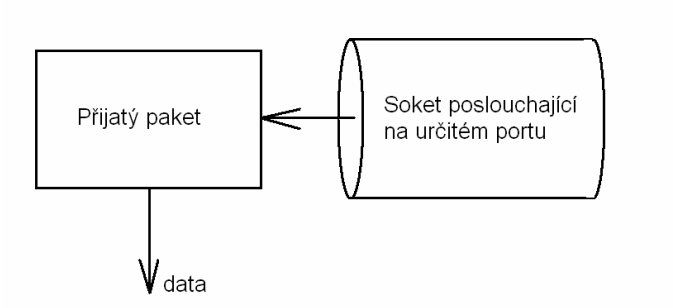
V Java SE je ve standardním aplikačním rozhraní určena pro práci v síti knihovna *java.net*, která obsahuje potřebné třídy. Pro komunikaci přes UDP nám postačí třídy *DatagramSocket* a *DatagramPacket.* Z názvů je poměrně patrné, že první zmíněná vytváří soket pro posílání paketů a druhá vytváří jednotlivé pakety.

Pro posílání dat je nejprve nutné vytvořit instanci třídy *DatagramSocket.* Pokud je jedno, z jakého portu budou data poslána, pak stačí vytvořit instanci bez určení portu a vždy při poslání paketu bude systémem nějaký port přidělen. Dalším krokem je vytvoření paketu, který budeme naplňovat a posílat. Zde už je nutné určit základní parametry paketu. IP Adresu na kterou se bude paket posílat, port na který má být paket doručen a hlavně data, která paket ponese. Ted už stačí pouze odeslat daný paket pomocí vytvořeného soketu.



Obr.1.1: Schéma odeslání paketu přes UDP

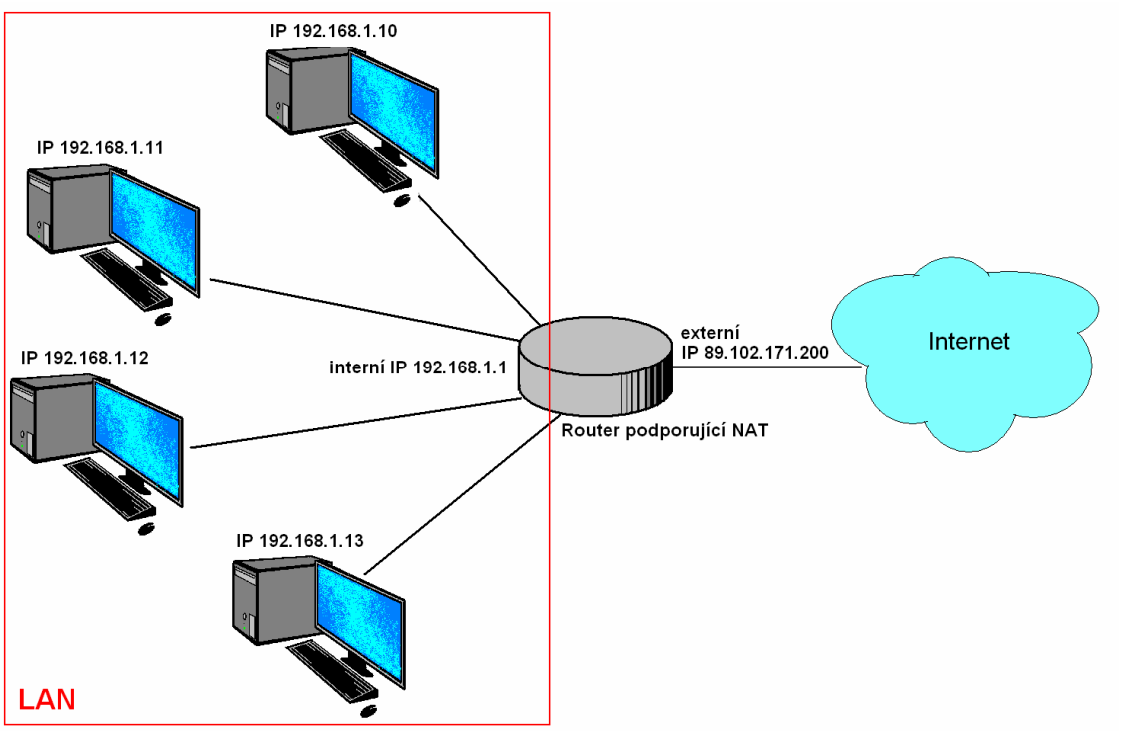
Obdobným způsobem se provádí i příjem dat. Nejprve se vytvoří instance paketu a potřebný soket. Soketu se tentokrát musí určit port, na kterém bude poslouchat. Pak se mu předá instance paketu, která se naplní, pokud je nějaký paket na tomto portu přijat a z něho se získají přijatá data.



Obr. 1.2: Schéma přijetí paketu přes UDP

V jazyce Java je díky API implementace UDP poměrně jednoduchá a nebudu se jejím popisem tedy dále více zabývat. A vzhledem k tomu, že jak port a adresa, tak i data se na paketu dají nastavit pomocí metod daného objektu, není tedy vůbec nutné znát přesné hodnoty rozložení jednotlivých bitů v UDP či IP hlavičce paketu či jejich délku.

Aplikace virtuální třídy využívají síťovou TCP/IP komunikaci, a tak se dřív nebo později při jejím vývoji musíme zabývat tématem NAT, tedy Network Address Translator. Pouze okrajově uvedu, že NAT zajišťuje na routeru oddělení vnitřní LAN sítě od vnější sítě. Kdy na vnitřní LAN síti jsou použity vlastní IP adresy s vlastní maskou a jako výchozí brána pro tuto síť ven do Internetu je právě onen router, který se o překlad těchto adres stará (viz obr. 1.3).



Obr. 1.3 : Příklad oddělení lokální interní sítě od vnější pomocí NATu.

Ovšem překonání této „bariéry“ v podobě překladače adres není kolikrát zrovna jednoduchou záležitostí. Tento překladač nepropustí pakety z vnější sítě pokud nebyla komunikace iniciována od některého klienta ze sítě vnitřní. Nastává tedy otázka, jakým způsobem přenést pakety putující mezi serverem Virtuální třídy a klientem, a také, jak rozpoznat kterému klientovi ve vnitřní síti patří. Jednou z možností, jak zajistit přenos UDP paketů, je právě ona inicializace ze strany klienta. Což jsem si otestoval na příkladu, kdy jsem vytvořil jednoduchý UDP server, který na žádost od klienta vrátil nějaká data. Což ovšem nejde implementovat v případě VT. Řešením by byla například implementace nějakého TCP tunelu pro UDP pakety, což už se ovšem vymyká zadání mé práce.

K tématu NAT jsem se vyjádřil záměrně i přesto, že má klientská aplikace nedokáže tuto bariéru překonat, protože jsem se snažil tento problém vyřešit, avšak zjistil jsem, že jeho řešení by opravdu bylo komplikované a přesahovalo by mé možnosti a také rámec zadání.

1.2.3 Multicast ve VT

V systémech VT se také někdy používá multicastového typu komunikace. Pro podporu tohoto typu komunikace je ovšem zapotřebí , aby všechny síťové prvky, přes které jsou klienti spojeni tento typ komunikace podporovaly a tedy i protokol s touto komunikací spojený. Hodně VT využívá mutlicastovou komunikaci pro audiovizuální přenosy. Tedy pro síťovou komunikaci, která nejvíce zatěžuje přenosové linky. Na druhou stranu spousta VT buď multicast nevyužívá vůbec nebo nabízí možnost přepnutí na unicast přenos. Navíc ve VT je asi vždy zapotřebí připojení k nějakému centrálnímu serveru, který se stará o řídící procesy a uchovávání informací v databázi. Ať už je to vlastní serverová aplikace s databází, nebo webový server.

1.2.4 Implementace videokonference ve VT

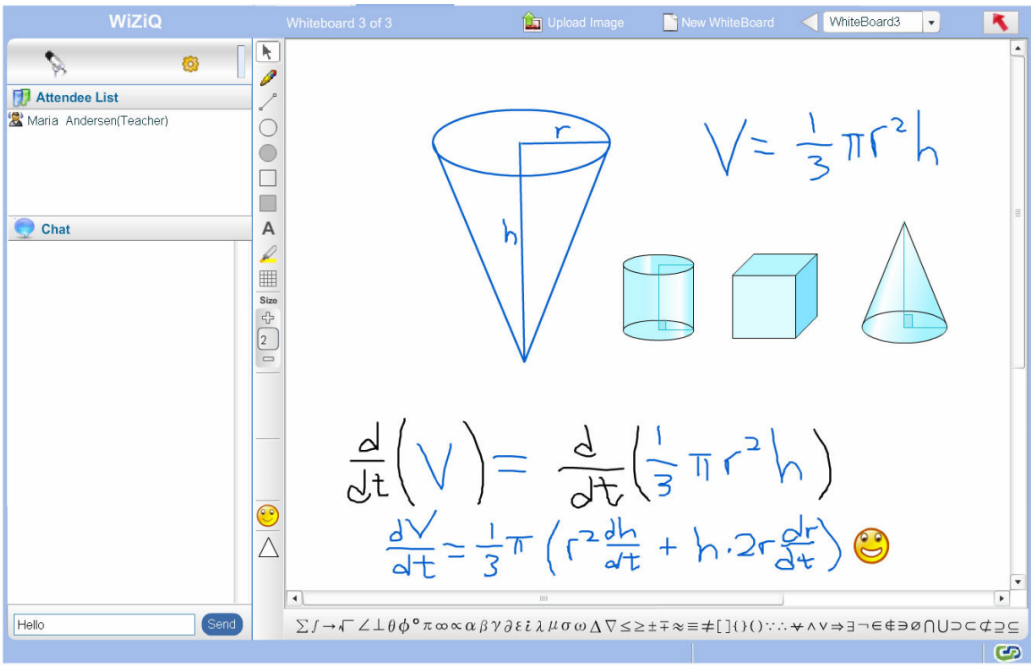
Pro to, aby spolu mohli dva koncoví klienti komunikovat také pomocí videa je zapotřebí samozřejmě, aby oba měli kameru, ze které bude aplikace získávat video data. Pak je zapotřebí získaný signál zakódovat pro přenos přes IP sítě. Pro videokonference se používá standardně protokol H.323, ovšem pokud by se jednalo o přenos pouze a jenom mezi danou aplikací, pak by se mohl prostě jen použít nějaký kodek pro video přenos (záleží na vývojáři) a jediným požadavkem by bylo, aby oba klienti měli stejnou klientskou aplikaci, se stejně nastaveným kódování a dekódováním videa. Je zapotřebí tedy obraz snímat v nějakém formátu a tento pak převést do streamu, z něhož se budou plnit pakety a posílat druhému uživateli. Přenos se standardně řeší přes transportní protokol UDP (User Datagram Protocol) s využitím aplikačního protokolu RTP (Real Time Protokol) pro zajištění dostatečné šířky pásma pomocí QoS (Quality of Servis). U druhého uživatele zas musí nastat opačný postup, kdy se tyto pakety opět budou vkládat do nějakého výstupního streamu, který bude skládat pakety do videosignálu promítaného do daného okna. V Javě lze využít pro tyto typy komunikace knihovny JMF (Java Media Framework) určené pro multimediální aplikace. Tímto způsobem by se tedy dala poměrně snadno vytvořit komunikace mezi dvěma koncovými uživateli.

Komplikace ovšem nastane při požadavku video spojení mezi více uživateli. Ne ve smyslu, že by jeden uživatel vysílal video ostatním, ale ve smyslu, že by každý uživatel vysílal svou kamerou snímané video každému dalšímu. Logicky se musí v takové situaci vyskytnout nějaký centrální bod, který bude rozdělovat pakety, přeposílat je ostatním a hlavně nějakým způsobem oddělovat od sebe, aby se chaoticky nepromíchaly. K tomuto účelu pro videokonference právě slouží hardwarová jednotka zvaná MCU (Multipoint Control Unit), která se právě stará o řízení videokonference při spojení většího množství uživatelů, což bezpochyby u systému VT může nastat.

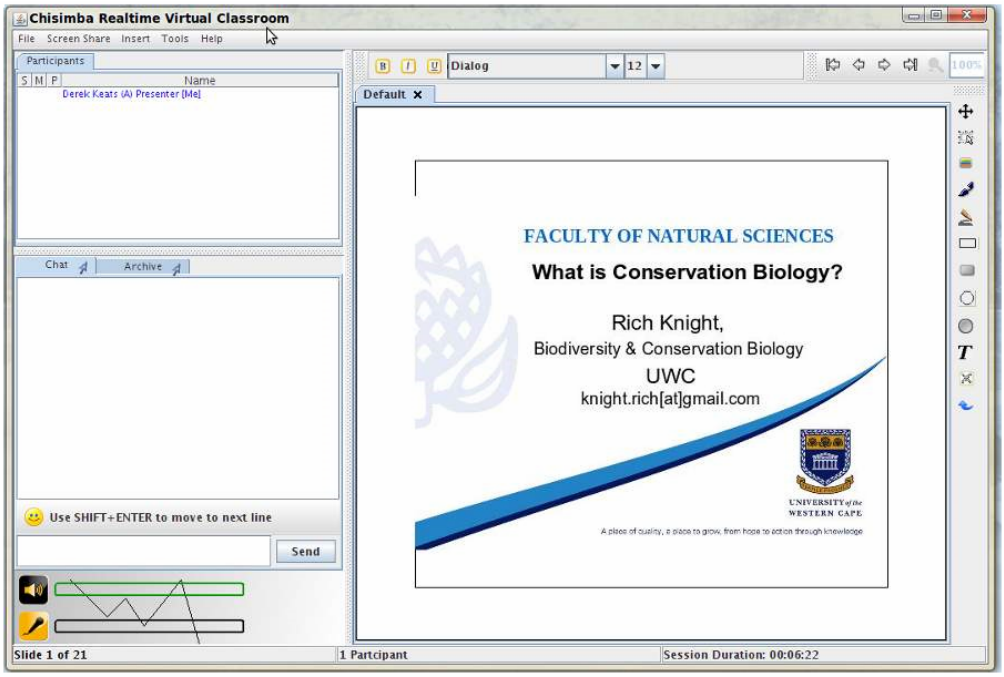
Tedy pokud bychom chtěli do VT implementovat video přenos pouze třeba od tutora ke všem ostatním uživatelům, tak by stačilo pouze použít postup popsaný výše pro spojení dvou klientů, pouze by se pozměnilo to, že pakety by byly posílány všem ostatním připojeným uživatelům. Pokud bychom však chtěli vytvořit VT, kde už by byla implementována videokonference více zdrojů, pak by se už musel buď vytvořit speciální software a implementovat na nějaký centrální server nebo využít přímo sofistikovaný hardware právě v podobě hardwarové jednotky MCU.

1.3 Ukázka produktů

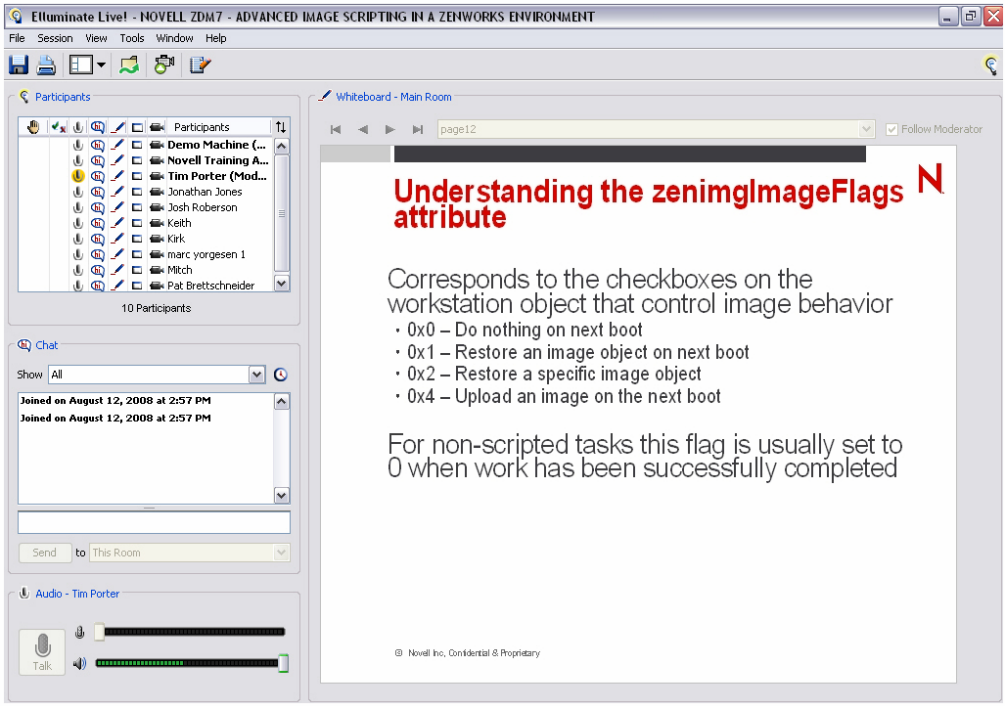
Zde bych chtěl uvést takovou ukázku toho, jak může běžná virtuální třída vypadat. Je mnoho komerčních i nekomerčních produktů. Já jsem vybral jako ukázku například dvě freewarové virtuální třídy. Jedna je součástí většího e-learningového systému Chisimba a druhá se jmenuje WizIQ. Vybral jsem je záměrně pro ukázku, protože Chisimba je vytvořena jako desktopová klientská aplikace v Javě a naopak WizIQ je čistě webově založená aplikace. Obě dvě podporují standardní prvky jakou jsou chat, whiteboard a audio nebo i video komunikace. Jako další poskytovatele virtuálních tříd bych mohl jmenovat například komerční Blackboard či Elluminate. Níže lze vidět ukázky některých z nich. Je vidět, že grafické rozhraní má ve všech ukázkách zhruba stejný styl a podporované funkce se také v základu příliš neliší. Teprve až propracování detailních funkcí jednotlivých nástrojů je to, co je jakýmsi ukazatelem kvality třídy. Takovým rozdílem může být například rozšíření možností kreslení ve whiteboard o nové funkce, či poskytnutí kvalitnějšího zvukového přenosu atp.



Obr.1.4 : Ukázka grafického prostředí čistě webové virtuální třídy WizIQ. Zdroj [online]: (http://tcmtechnologyblog.blogspot.com/2007_11_01_archive.html)



Obr.1.5: Ukázka grafického prostředí virtuální třídy Chisimba, jakožto desktopové Java aplikace. Zdroj [online]: (http://www.dkeats.com/index.php?module=blog&action=viewblogbytag&tag=%23chisimba &userid=1563080430)



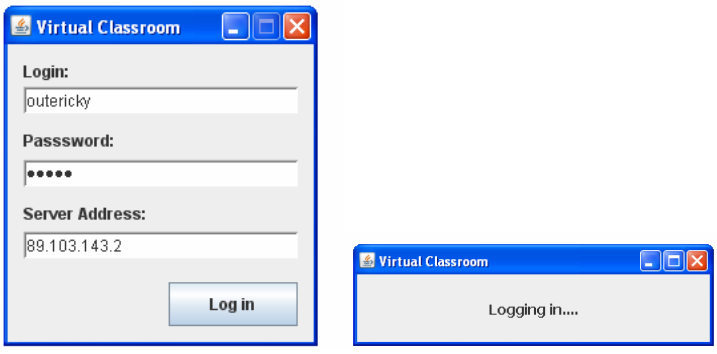
Obr. 1.6: Ukázka grafického prostředí virtuální třídy Elluminate, jakožto desktopové aplikace. Zdroj [online]: (http://www.novell.com/connectionmagazine/2009/03/img/att2.jpg)

2 Úvod k vlastní realizaci VT

Součástí zadání mé práce je hlavně vytvoření návrhu a realizace vlastní VT pomocí jazyku Java. Jak bylo popsáno, tak virtuální třída by měla obsahovat hlavně tři základní nástroje a těmi jsou chat, whiteboard a nějaký typ audio komunikace. Já jsem se snažil implementovat pokud možno dostatečné množství nástrojů pro podporu vzdálené výuky. Další kapitoly jsou tedy věnovány podrobnému popisu mnou vytvořené VT.

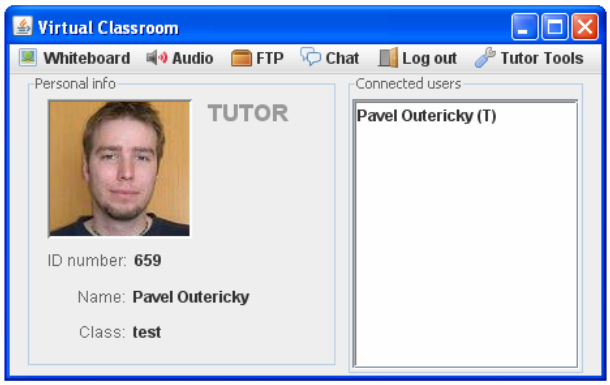
Jak bylo řečeno v kapitole *Softwarová řešení VT,* je více možností jak VT realizovat. Já jsem si vybral pro svou práci alternativu desktopové aplikace komunikující se serverovou aplikací. K vývoji obou jsem použil prostředí volně dostupného vývojového nástroje NetBeans. A jak server, tak i klientská aplikace jsou napsány v jazyce Java 2 SE. A tedy ke správné funkci je zapotřebí, aby měl daný počítač nainstalované prostředí (virtuální) pro spouštění Java aplikací (Java Runtime Environment).

Klient tedy má na svém počítači nainstalovanou klientkou aplikaci, kterou v danou dobu spustí. Poté zadá potřebné údaje k přihlášení, kterými jsou login, heslo a adresa serveru (adresa serveru je zde nastavena tak, že se musí explicitně zadávat – což je však spíše z testovacích účelů. Adresa by se dala implicitně nastavit v kódu programu, aby ji uživatel nemusel pokaždé zadávat). V programu je využit externí soubor, do kterého se ukládají údaje naposledy přihlášeného uživatele. Klient tak nemusí pokaždé zadávat své údaje znovu, ale stačí zadat pouze heslo.

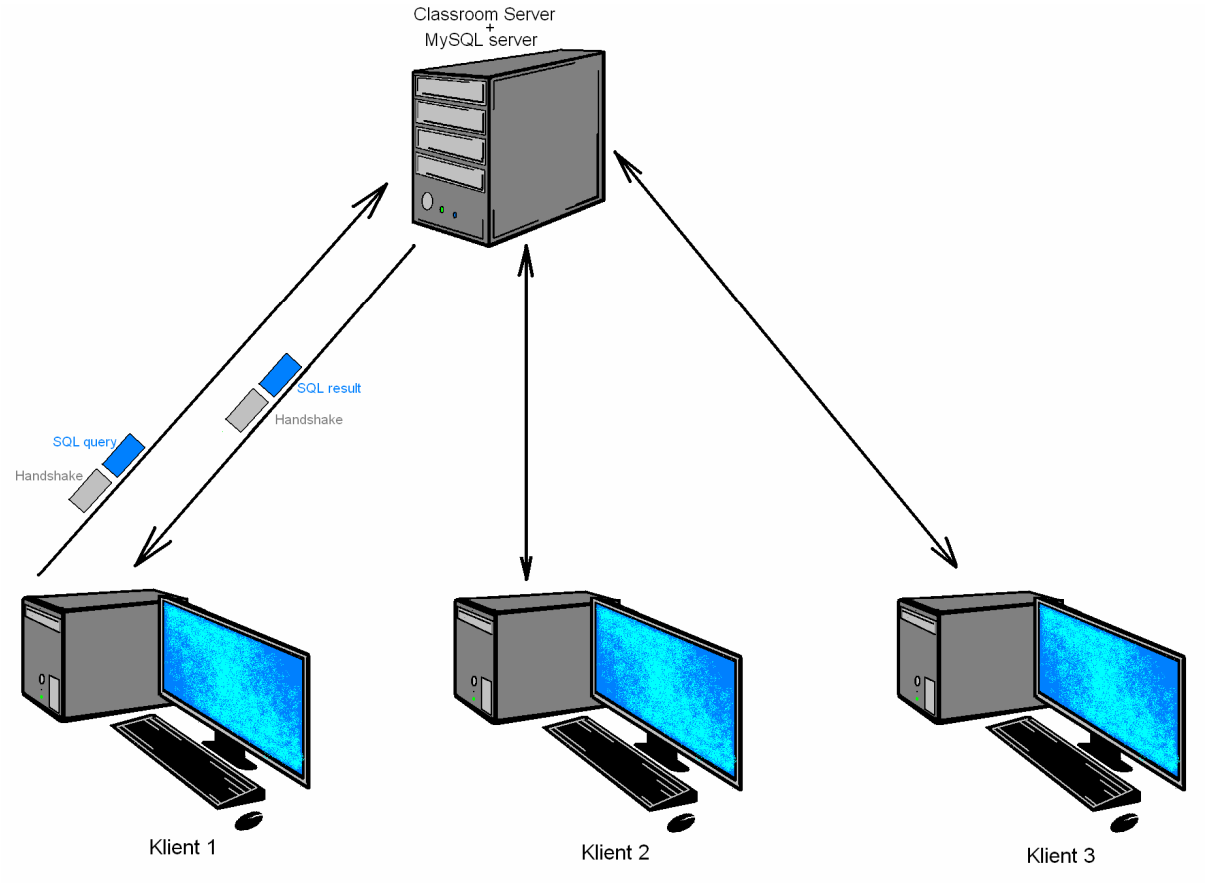


Obr. 2.1: Přihlašovací tabulka zobrazená po spuštění aplikace (vlevo) a informace o přihlašování, která je zobrazena po potvrzení údajů, než se objeví hlavní okno a klient je přihlášen.

Po zadání a potvrzení potřebných údajů jsou tyto poslány na adresu serveru, kde běží také databázový MySQL server a z něj jsou získána data o uživateli (viz kapitola *Databáze studentů*). Poté je poslán tzv. handshake paket na ten samý server, ale ten je pro serverovou aplikaci Classroom Server, se kterou se provede handshake a klient je zařazen do seznamu přihlášených uživatelů (viz kapitola *Řídící komunikace*). Nyní může začít komunikovat s ostatními přihlášenými uživateli a využívat funkce programu.



Obr. 2.2: Grafické rozhraní klientské aplikace po úspěšném přihlášení.



Obr. 2.3: Ukázka komunikace při přihlašování klienta 1.

3 Databáze klientů

3.1 Možnosti

Každý systém tohoto typu potřebuje nějakým způsobem uchovávat údaje o klientech, zde v tomto případě údaje o studentech a tutorech. Logicky se tedy nabízí využití nějakého typu databáze a databázového serveru. Při výběru databáze máme více možností. Nejrozšířenějšími jsou v dnešní době asi databáze relační. Relační databáze se vyznačují především tím, že jsou relativně jednoduché, snadno pochopitelné a díky tomu rozšířené a mezi programátory a vývojáři informačních systémů velmi oblíbené. [17] Nejznámější databázové systémy jsou asi MySQL, PostgreSQL nebo například Oracle. Pro mou práci jsem si vybral volně dostupnou freeware verzi těchto databází a tou je MySQL.

Nejprve bylo tedy hlavně nutné nainstalovat nějakou verzi tohoto databázového serveru. Na Internetu je volně k dispozici několik verzí tohoto typu serveru. Já jsem si vybral MySQL Server 4.1 dostupný na adrese [www.mysql.com].

Server byl nainstalován a spuštěn. Nastavení serveru jsem zvolil jako základní všeobecnou databázi a typ , který běží na počítači vývojáře a počítač tedy není dedikován pouze tomuto databázovému serveru.

K vytvoření, editaci a také čtení z databáze je možné použít základního klienta ve formě příkazového řádku, který ovšem samozřejmě nenabízí příliš komfortní prostředí. K editaci SQL databází existuje řada podpůrných uživatelsky přívětivějších aplikací, z nichž mnohé jsou zdarma volně dostupné. Jednou z takových aplikací je právě SQLyog (viz obr. 3.1), který jsem zvolil k práci s mou databází já, abych mohl jednoduše vytvořit testovací bázi s několika uživateli.

3.2 Obsah databáze

Je mnoho údajů, které by se dali v rámci systému virtuální třídy do databáze ukládat. Mně zde šlo hlavně o to, vytvořit základní bázi dat, ze které budou moci studenti či tutoři načíst své osobní údaje či údaje o jiných studentech. A implementovat tak prostě možnost aplikace komunikovat s nějakou databází. Databáze tedy obsahuje jednu tabulku s názvem „studs“, ve které jsou obsažena data o studentech i tutorech.

Zvolil jsem tyto základní údaje o uživatelích:

„Id“ - identifikační a tedy jedinečné číslo studenta či tutora. Databázový typ smallint.

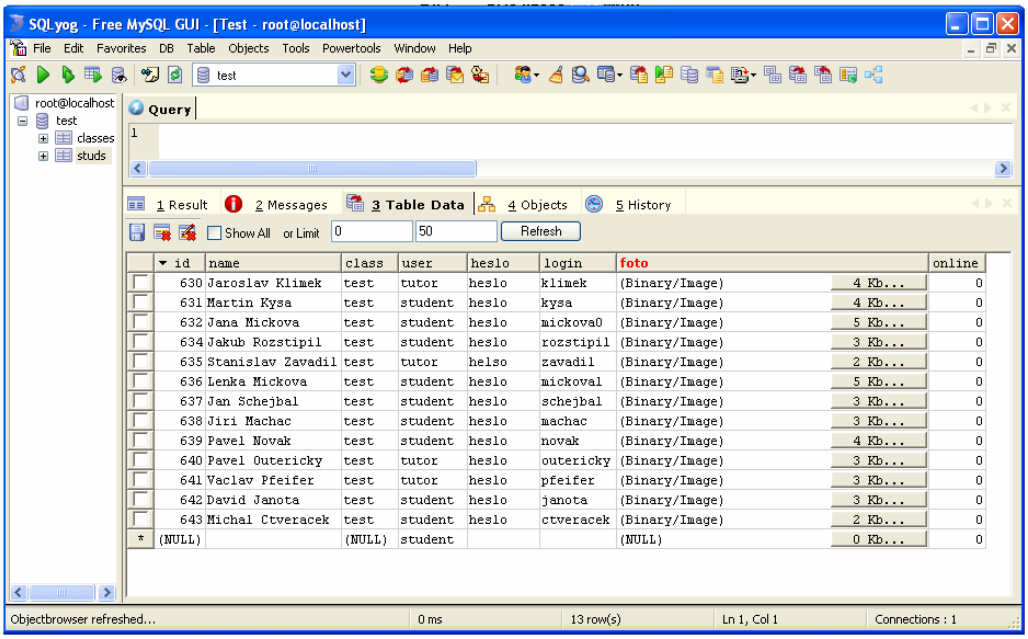
„name“ - kolonka pro jméno studenta. Typ varchar, maximálně 50 znaků.

„class“ - třída, do které je student zařazen. Typ varchar.

„user“ - určení zda je uživatel tutor či student. Typ varchar. Nepoužil jsem boolean hodnotu z důvodu možného rozšíření na další typy uživatelů.

„password“ - heslo uživatele. Typ varchar. Pro zabezpečení hesla by zde dala použít šifrovaná hodnota získaná pomocí funkcí MD5 či SHA-1.

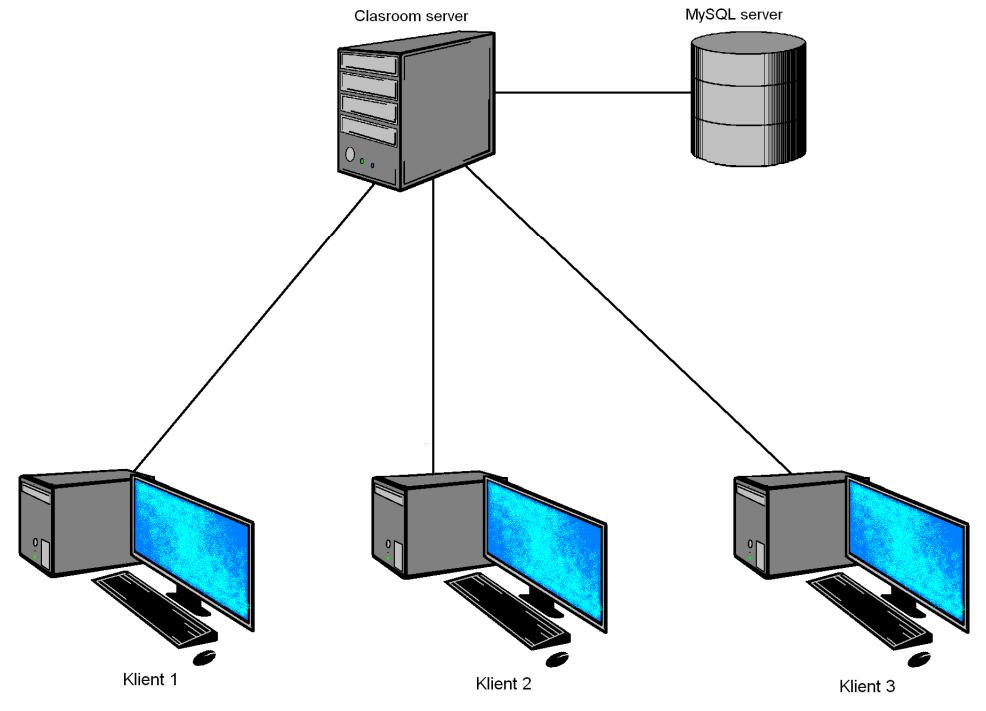
„ login“ - unikátní přihlašovací jméno studenta. Používá se příjmení s případným číselným rozšířením.



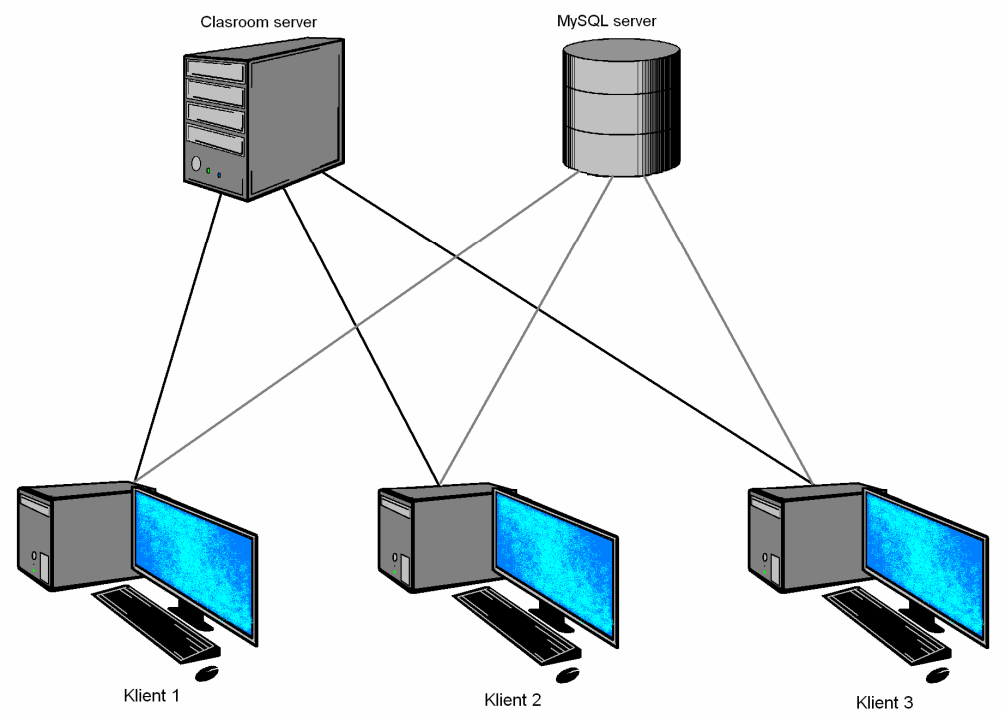
Obr. 3.1: Ukázka tabulky databáze v prostředí SQLyog.

3.3 Implementace v kódu a komunikace s databázovým serverem

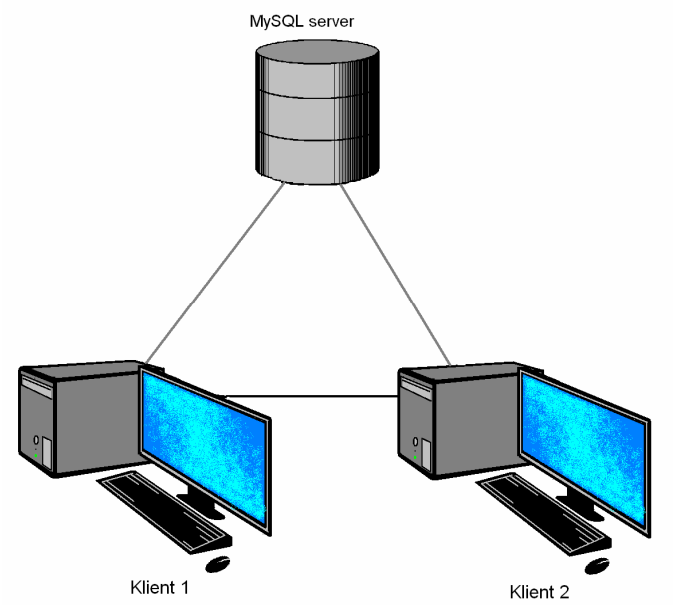
Komunikaci s databázovým serverem iniciuje klientská aplikace. Jsou dvě základní možnosti, jak by komunikace mohla probíhat. Buď k databázi přistupuje pouze serverová část aplikace (obr. 3.2), která získá z databáze potřebná data a pošle je klientovi nebo k databázi může přistupovat přímo klient a získat si tak data přímo z databázového serveru sám (obr. 3.3) První možnost má samozřejmě jisté bezpečnostní výhody a druhá možnost má zase výhodu v tom, že serverová aplikace nemusí vykonávat další režii v podobě komunikace s databází. V mé práci jsem vytvořil komunikaci přímo klienta s databází a to hlavně z důvodů, že první jsem vyvíjel klientskou aplikaci, kdy mohli komunikovat dva uživatele navzájem mezi sebou a oba měli přístup k databázi (obr. 3.4) a pak teprve jsem vyvíjel serverovou část pro spojení více uživatelů.



Obr. 3.2: Komunikace s databází přes serverovou aplikace.



Obr. 3.3: Přímá komunikace klientských aplikací s databázovým serverem.

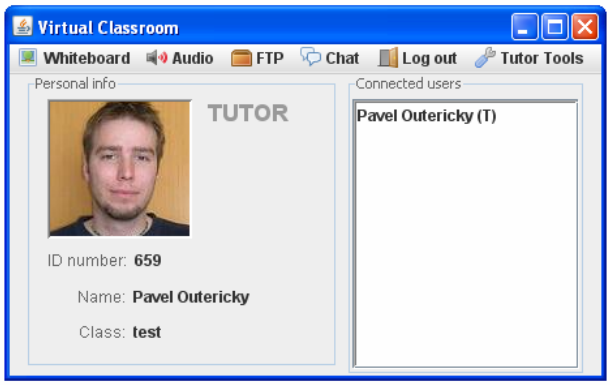


Obr. 3.4: Komunikace dvou klientských aplikací mezi sebou a s databázovým serverem.

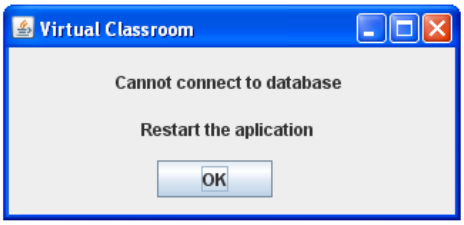
Pro implementaci v klientské aplikaci byla zapotřebí knihovna, která vytvoří podporu pro komunikaci s SQL databází, takzvaný konektor. Pro Java aplikace je k dispozici volně dostupný Connector/J – JDBC driver. Tato knihovna obsahuje funkce pro práci s MySQL databází. Já jsem ji tedy naimportoval do svého projektu a použil ve vlastní třídě, která se pak o komunikaci s databází stará.

3.3.1 Třída DBCommunication

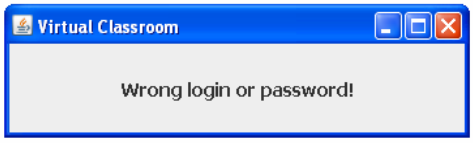
Po spuštění aplikace a načtení uživatelem zadaných údajů se vytvoří nová instance třídy *DBCommunication* [příloha A.1], které se již v konstruktoru předá adresa databáze, se kterou se má spojit. Po úspěšném spojení je vyvolána nad touto instancí metoda pro přihlášení, které se předají dva parametry, login a heslo, tím se z databáze získají pomocí SELECT příkazu a porovnání loginu a hesla data zadaného uživatele, která jsou pak dále používána. Získá se identifikační číslo, jméno, název třídy do které student (tutor) patří, pak údaj, jestli je to student nebo tutor a také jeho fotografie. V případě, že v databázi nebyl nalezen žádný uživatel daného loginu a hesla, je klientovi zobrazena informační zpráva. Stejně tak je zobrazena informační zpráva pokud se nelze vůbec spojit s databázovým server a danou databází.



Obr. 3.5: Grafické prostředí s načtenými daty z databáze.



Obr. 3.6: Informace o nedostupnosti databáze.



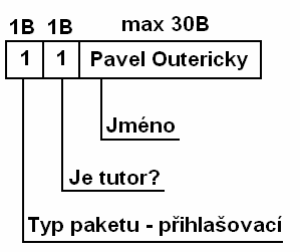
Obr. 3.7: Informace o špatném zadání údajů.

4 Řídící komunikace

Pro řízení přístupu, přihlašování a odhlašování klientů, přepínání mluvčího (viz kapitola *Audio Komunikace*) a další záležitosti bylo zapotřebí vytvořit vlastní komunikační řídící protokol mezi serverem a klienty, který bude tyto záležitosti řídit. Na serveru tedy běží vlákno, které poslouchá na daném portu a přijímá řídící pakety. Po přijetí paketu zjistí, jaký je to typ řídícího paketu a podle toho provede příslušnou operaci. Stejně tak na straně každého klienta běží vlákno, které na tomto portu poslouchá a přijímá případné řídící pakety od serveru. Na obou stranách se o tuto režii stará třída *HandShake* [příloha A.2] . Níže jsou tedy popsány možné řídící operace a jejich implementace v protokolu.

4.1 Přihlášení uživatele

Zde bude asi nejlepší použít k popisu přímo příklad. K přihlášení použiji svůj login: „outericky“ a heslo: „heslo“ (jak je nastaveno v databázi), po potvrzení údajů v databázi jsou klientské aplikaci doručeny informace o mé osobě. Tedy celé jméno, fotografie, třída do které patřím, identifikační číslo a zda jsem tutor nebo student. Po obdržení těchto údajů pošle aplikace řídící paket směrem k serveru na daný řídící port. Paket obsahuje jméno a typ uživatele a samozřejmě informaci o tom, že je to přihlašovací paket (obr. 4.1) Server tento paket přijme, zjistí, že se jedná o přihlašovací paket a vytáhne si z něj ono jméno a typ klienta a také jeho adresu, odkud byl paket poslán. A vytvoří novou položku klienta v seznamu klientů. A nyní se dostáváme k seznamu klientů na serveru. O toto se starají dvě třídy.



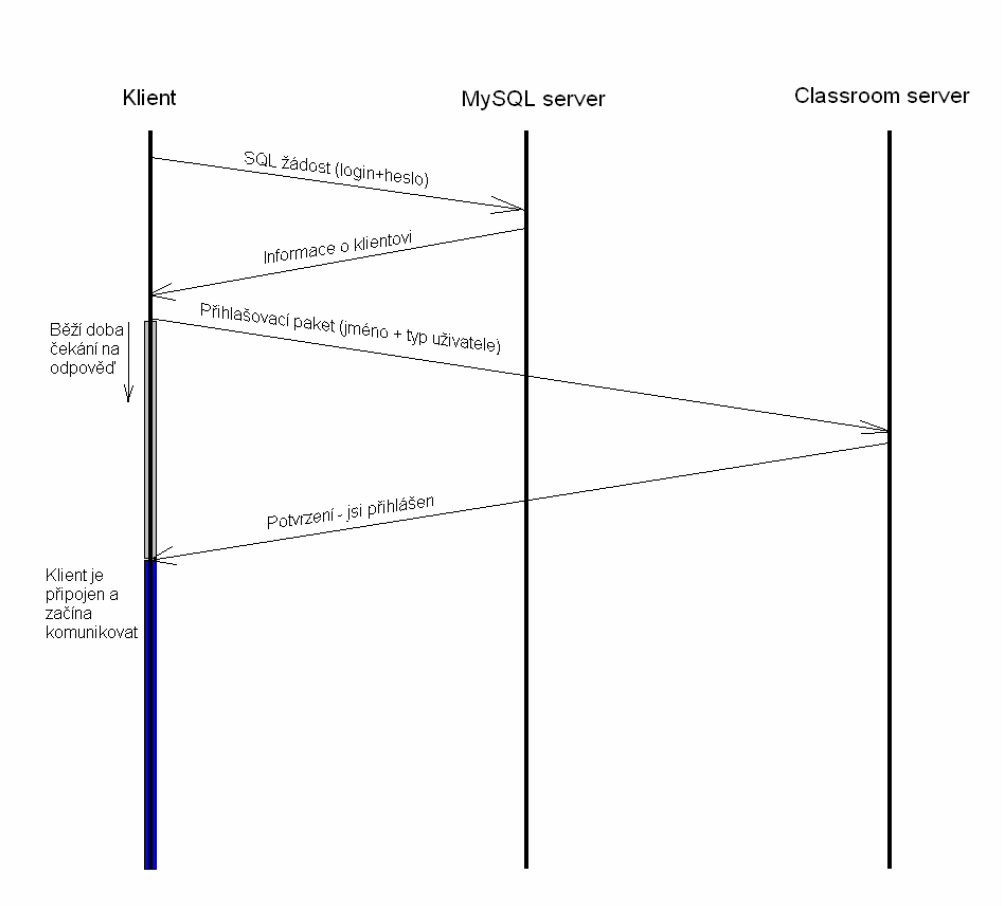
Obr. 4.1: Ukázka obsahu přihlašovacího paketu.

4.1.1 Třída Client

Tato třída definuje jednoho klienta [příloha B.2]. Každá instance, tedy každý klient má tři základní parametry *String name, InetAddress address, boolean isTutor*. Tedy jméno, adresu a určení, jestli je tutor nebo student. Třída obsahuje metody pro získaní těchto parametrů z daného klienta, či nastavení těchto parametrů, tedy klasické „set“ a „get“ metody.

4.1.2 Třída ClientList

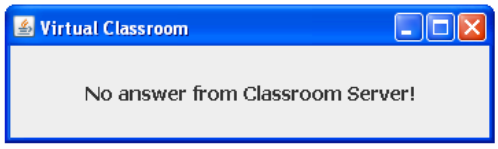
Definuje seznam klientů [příloha B.2]. V této třídě je statické pole objektů typu *Client*. Třída definuje metody pro vložení klienta na nejbližší volnou položku v seznamu, získání klienta podle indexu v seznamu, smazání daného klienta a taky metodu, která vrátí seznam přihlášených klientů jako pole bytů naformátované dle protokolu a připravené k odeslání v paketu uživatelům pro obnovení jejich seznamu o přihlášených uživatelích.



Obr. 4.2: Postup při přihlašování klienta.

4.1.3 Potvrzování při přihlášení

Při přihlášení klienta na server také mimo jiné server pošle danému klientovi potvrzovací paket o tom, že byl úspěšně přihlášen a klient začne běžně komunikovat a přijímat pakety. Pokud se tak ovšem nestane a klient nedostane od serveru odpověď do určitého daného intervalu, pak pošle přihlašovací paket na server znova. Časový interval, kdy klient čeká na odpověď od serveru o úspěšném přihlášení, se s každým pokusem zvětšuje (z důvodu možného zatížení serveru zkouší klient počkat déle). V aplikaci je nastaveno pět pokusů. Po proběhnutí pátého pokusu a vypršení čekacího intervalu se klientovi zobrazí informace o nedostupnosti serveru (obr. 4.3).



Obr. 4.3: Informace o nedostupnosti serveru.

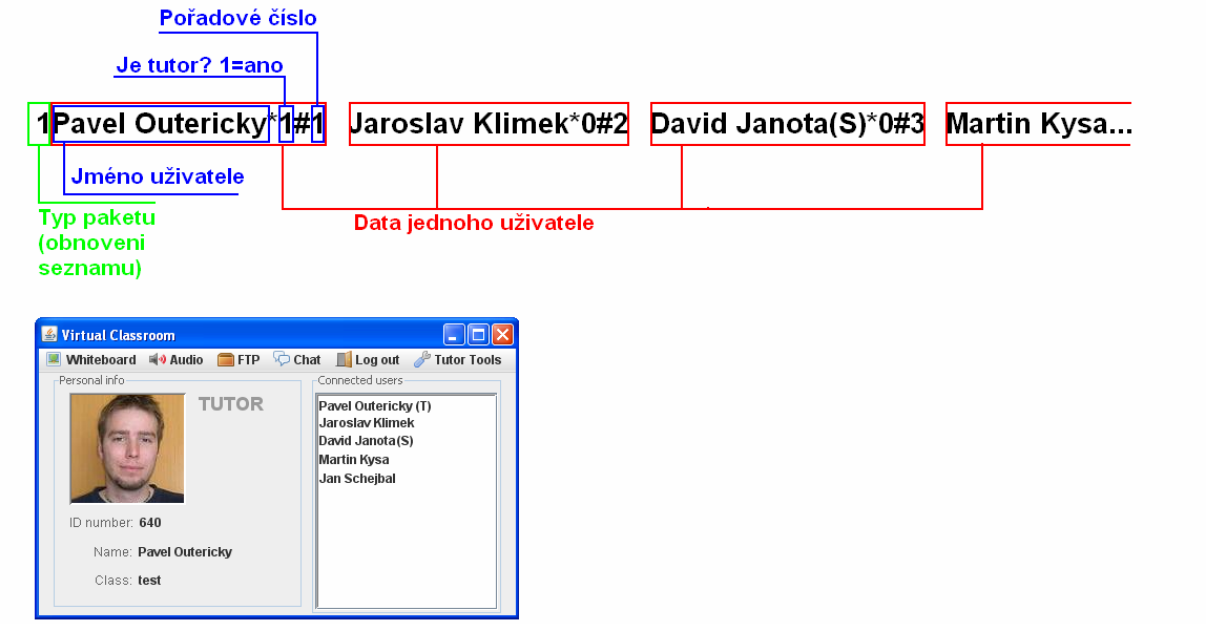
4.2 Odhlášení uživatele

Stejně jako je v protokolu přihlašovací paket, tak je samozřejmě i paket odhlašovací. Pokud uživatel zavře svou aplikaci nebo použije možnost „Log out“, pak se ještě před uzavřením aplikace pošle paket pro odhlášení. Pokud server obdrží tento paket, zjistí, že je odhlašovací, pak podle adresy najde klienta ve svém seznamu a smaže ho. Klient nečeká na potvrzení od serveru. Pokud by nastala situace, že paket není doručen a klient zůstane přihlášený na serveru, což je samozřejmě nežádoucí stav, pak tuto situaci za pár vteřin vyřeší kontrola přihlášených klientů (viz níže).

4.3 Obnovování seznamu klientů

Jak jsem již výše zmínil, tak třída *ClientList* obsahuje metodu, která vrátí seznam klientů naformátovaný dle protokolu přímo pro vložení do paketu (obr. 4.4) Při každém přihlášení klienta, odhlášení klienta či změně mluvčího dojde tedy ke změně seznamu přihlášených uživatelů a musí se tedy informovat klienti o změně.

K tomu právě slouží paket pro obnovení seznamu, který je odeslán všem uživatelům pokaždé, nastane-li některá ze zmíněných situací (přihlášení či odhlášení uživatele nebo změna mluvčího – probráno níže). Každý klient tento paket přijme, zpracuje a obnoví si podle něj zobrazený seznam připojených uživatelů. Vzhledem k tomu, že je pro tuto komunikaci použito nepotvrzované UDP komunikace, tak by se mohlo stát, že paket nebude doručen a z toho důvodu je pro jistotu na serveru nastaveno cyklické rozesílání těchto paketů o seznamu uživatelů. Interval jsem nastavil na každých osm sekund. Každých osm sekund se tedy rozešle všem klientům seznam přihlášených uživatelů,

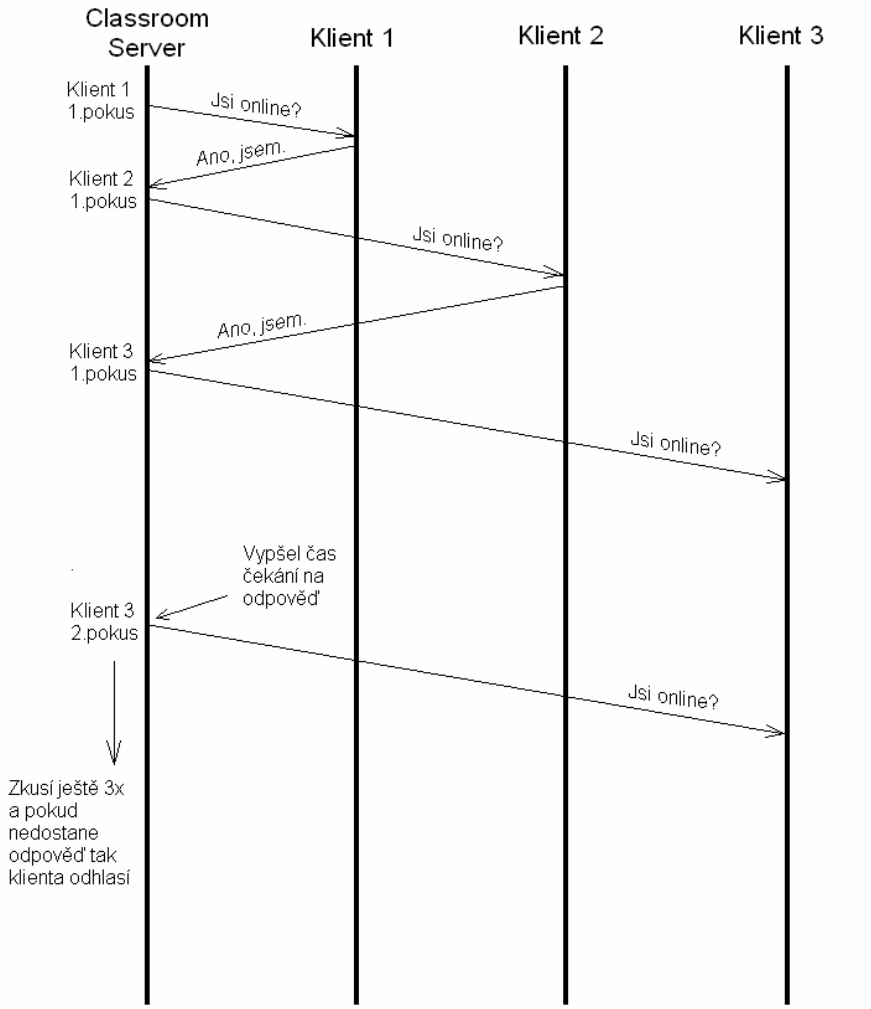


*Obr. 4.4: Obsah paketu se seznamem uživatelů poslaný od serveru klientům a výsledný obsah seznamu přihlášených uživatelů dle ukázkového paketu. V obrázku jsou popsánu jednotlivá pole. Jako oddělovače jsou požity znaky * a #. Je zde vidět, že klient Outericky je tutor a klient Janota je právě nastaven jako mluvčí (k jeho jménu se na serveru přidá „(S)“).*

4.4 Kontrola přihlášených klientů

Vždy může nastat situace, že klient se nečekaně odpojí, nebo se mu stane nějaká závažná chyba na počítači či například u něj vypne elektřina, a proto je zapotřebí tuto situaci nějak ošetřit také na serveru. Z toho důvodu jsem implementoval kontrolu přihlášených uživatelů. Jedno z běžících vláken na serveru se stará o to, že každých pět sekund rozešle kontrolní paket.

Vždy vezme jednoho uživatele ze seznamu, pošle mu kontrolní paket, počká daný čekací interval a pokud uživatel pošle zpět tzv. „IamOnline Packet“ a server ho přijme, pak vezme dalšího uživatele ze seznamu a opět mu pošle kontrolní paket atd. Pokud server nedostane od uživatele odpověď, pak to zkusí ještě čtyřikrát a pokud ani po těchto dohromady pěti pokusech nedostane žádnou odpověď od klienta, tak ho smaže a vezme opět dalšího klienta ze seznamu (obr. 4.5).



Obr. 4.5: Průběh kontroly přihlášených klientů, kdy klient 3 neodpovídá.

4.5 Řídící pakety pro přepínání mluvčího

Kromě řídících paketů pro správu uživatelů jsou zde ještě pakety pro přepínání mluvčího. Systém přepínání mluvčího je popsán v kapitole o audio komunikaci.

4.6 Shrnutí řídících paketů

Každý řídící paket má tedy svůj přesně daný typ, aby bylo jasné o jaký paket se jedná a jak se má zpracovat (viz Tab. 4.1).

Tabulka 4.1: Přehled řídících paketů

Typy řídících paketů	
Pakety od serveru ke klientovi	
0	Dotaz jestli je klient online
1	Obnovení seznamu připojených klientů
2	"Mute" paket - pokud je uživatel aktivní mluvčí tak přestane vysílat
3	"UnMute" paket - uživatel začne vysílat audio pakety
Pakety od klienta k serveru	
0	Přihlašovací paket
1	Odhlašovací paket
2	"IamOnline" paket - potvrzení na dotaz jestli je klient připojen
3	Paket nesoucí informace o změně mluvčího

5 Posílání textových zpráv - chat

Dle mého názoru asi nejzákladnější funkce pro tento systém je možnost okamžité psané komunikace mezi všemi uživateli, tedy jakýsi chat či jak se říká instantní posílání zpráv. V mé virtuální třídě je tedy tato možnost komunikace samozřejmě také implementována a v dalších řádcích ji budu nazývat „chat“.

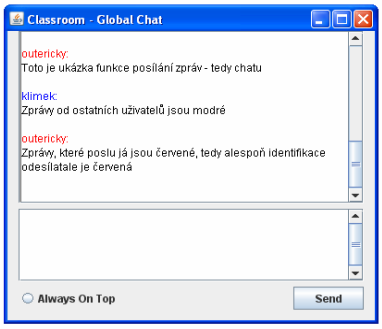
Hlavní okno grafického rozhraní nabízí v menu možnost otevření okna „Chat“. Po zvolení této položky se uživateli otevře okno pro chat, které obsahuje dva základní textové panely nad sebou, z nichž do spodního lze psát zprávu a v horním se zobrazují přijaté zprávy.

Zprávy jsou posílány vždy tak, že obsahují identifikaci uživatele, který ji poslal a textový obsah zprávy. Jako identifikace je zde použit login uživatele, který je v tomto systému nastaven jako příjmení uživatele bez diakritiky s případným číselným rozšířením, jak bylo uvedeno v části o komunikaci s databází. Dal by se samozřejmě použít jakýkoliv identifikátor, já jsem zvolil onen login. Dále pro lepší přehlednost jsou zprávy barevně odlišeny a to tak, že login zprávy od daného uživatele je červeně a login ve zprávě od ostatních, tedy vzdálených uživatelů, je modře, jak je ukázáno na obr. 5.1.

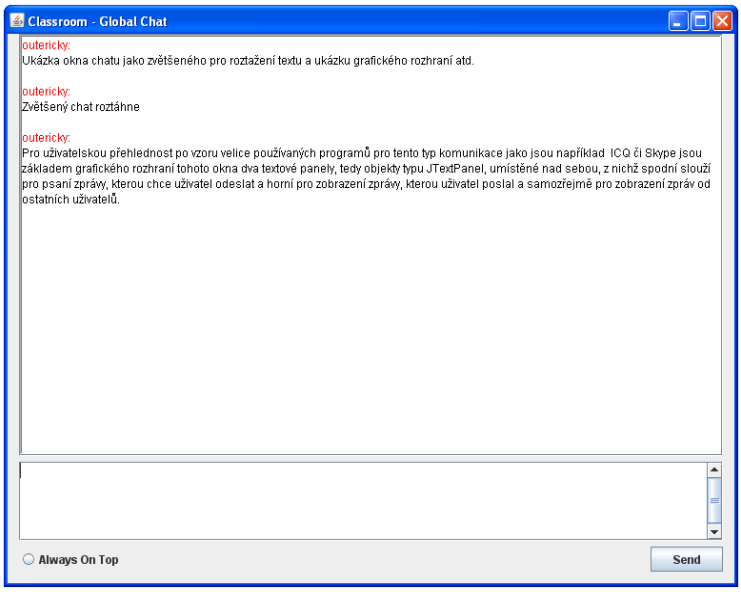
Napsanou zprávu je možno odeslat dvěma způsoby. A to buď klávesou CTRL nebo kliknutím na „Send“ tlačítko. Klávesu enter jsem k odeslání záměrně nepoužil, aby se dalo ve zprávě použít také odřádkování.

V okně chatu je také *radiobutton* „Always on top“, který nastaví okno chatu tak, aby bylo vždy nahoře (překrývalo ostatní otevřená okna na obrazovce) a tedy viditelné. Tuto funkci jsem implementoval hlavně z důvodu současného využití okna whiteboard (popsáno v dalších kapitolách) a chatu (obr. 5.3). Aby bylo stále vidět, jestli někdo z přihlášených klientů něco podotýká či se na něco ptá právě pomocí chatu.

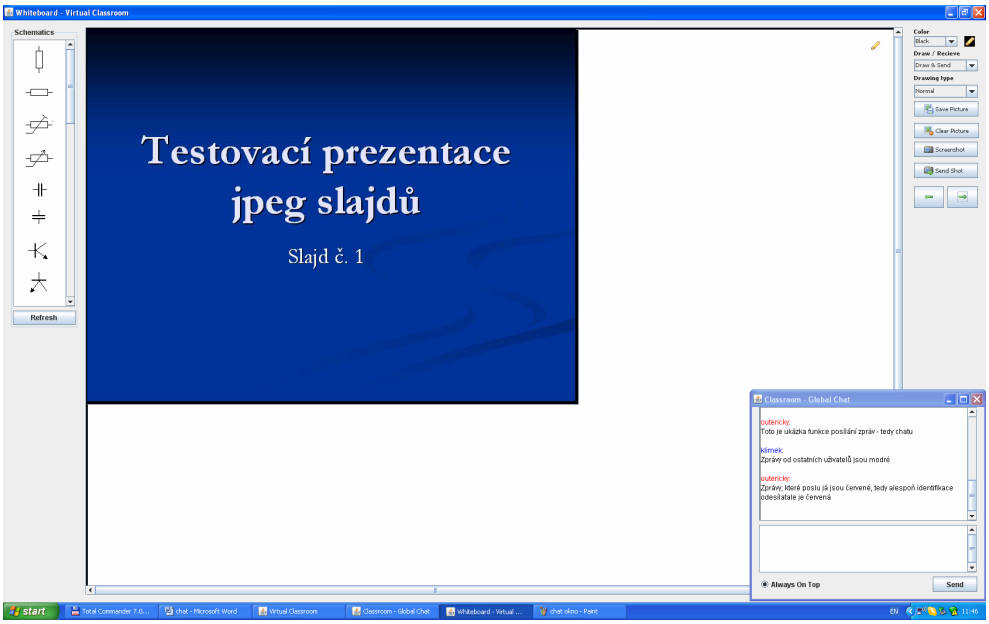
Ke grafickému rozhraní bych jestě dodal, že na změnu velikosti okna chatu reaguje horní panel vertikální i horizontálné a spodní panel pouze horizontálně.



Obr. 5.1: Okno chatu ve standardní velikosti.



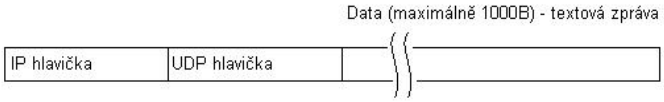
Obr. 5.2: Okno chatu zvětšené.



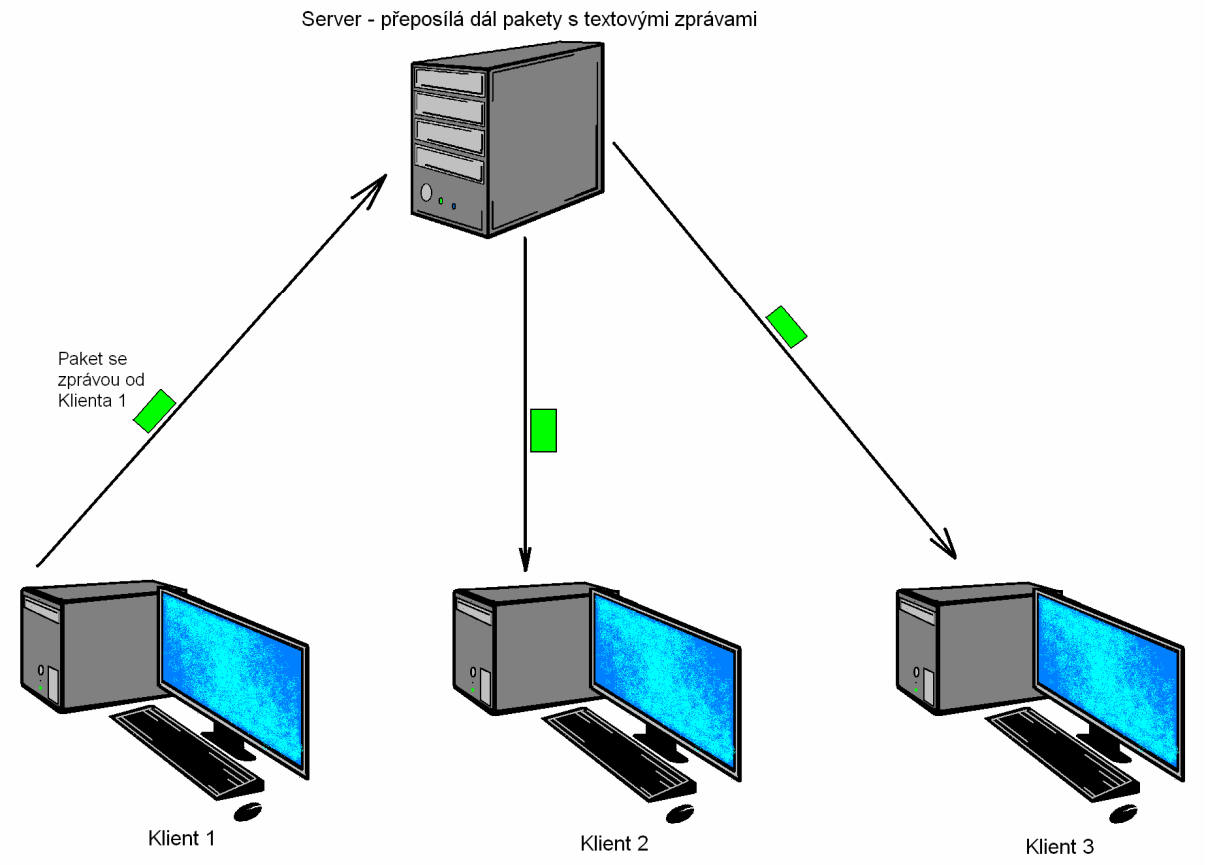
Obr. 5.3: Současné používání whiteboard a chatu, kdy chat je vždy viditelný.

5.1 Síťová komunikace pro chat

Síťová komunikace je zde řešena pomocí protokolu UDP a posílání datagramových paketů. Chat je dělaný tak, že přes server komunikují mezi sebou všichni připojení účastníci zároveň. Pro obsluhu této komunikace na straně klienta slouží třída *GlobalChatCommunication* [příloha A.3]. V této třídě jsou dvě hlavní statické metody pro obsluhu, z nichž jedna se stará o posílání zpráv a druhá o příjem. Zprávy jsou zapouzdřovány do paketu tak, že vždy jedna zpráva je jeden paket. Maximální velikost zprávy je zde nastavena na tisíc znaků (obr. 5.4).



Obr. 5.4: Obsah chat paketu.



Obr. 5.5: Schéma chat komunikace mezi serverem a klienty.

6 Audio komunikace

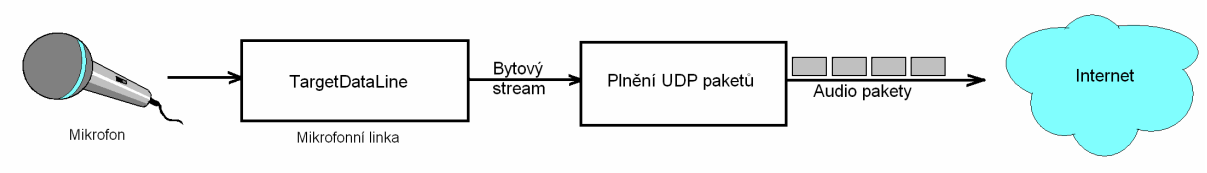
Systém komunikace by se dal navrhnout několika způsoby. Například, že budou komunikovat všichni společně jako při audio konferenci, či že bude moci mluvit každý s každým zvlášť nebo, že bude mít možnost mluvit vždy pouze jeden člověk. Každá z těchto možností má svou úroveň náročnosti co se týče implementace a dle mého názoru by samotné možnosti a způsoby implementace audio komunikace přes IP sítě vydaly na téma k diplomové práci. Vzhledem k tomu, že zadaní mé práce je vytvoření celého systému virtuální třídy, tak jsem implementoval i audio komunikaci.

6.1 Zpracování audio vstupu a odeslání

Prvním úkolem pro audio komunikaci je získat nějakým způsobem zvuk z připojeného mikrofonu a ten zpracovat do takové formy, aby se dal poslat dál. K tomuto účelu slouží v jazyce Java knihovna *AudioSystem*. Tato knihovna obsahuje možnosti pro práci se zvukovými zařízeními, které jsou na daném počítači k dispozici. Umožňuje získaní vstupní linky mikrofonu či výstupní linky pro reprodukci a samozřejmě přijímaní či posílání zvukových dat na tyto linky. Lze s touto knihovnou také měnit datové formáty zvuku atp. V tuto chvíli je pro mne však nejdůležitější získání linky mikrofonu.

1) Určíme formát, v jakém se mají data ze vstupní linky přijímat (použil jsem běžný formát pro přenos hlasu - hloubka vzorku 8 bitů, vzorkovací frekvence 8kHz, PCM - tedy jako kodek G.711).
2) Získáme vstupní linku mikrofonu, které se předá vytvořený formát zvuku.
3) Vytvoříme vstupní audio stream z dané linky.

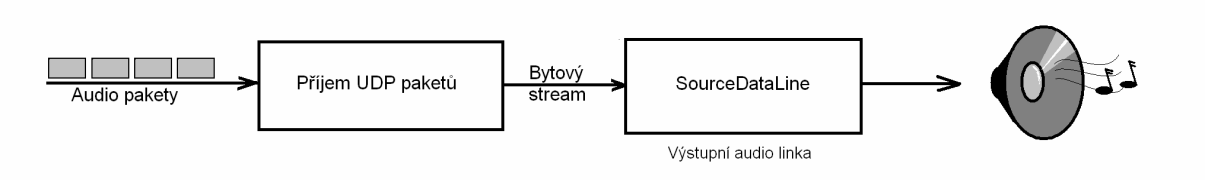
Pak se postupně snímá zvuk ze vstupní linky. Vždy se načte „kousek“ zvuku do bufferu (pole bytů) a ten se pošle v UDP paketu ven. V aplikaci tedy běží pro obsluhu audio komunikace dvě vlákna. Jedno vlákno načítá zvuk z vytvořeného mikrofonního streamu a posílá data po paketech ven a druhé vlákno přijímá tyto pakety na zadaném portu, na kterém poslouchá. Pakety zvuku nejsou žádným způsobem potvrzovány, jestli byly doručeny nebo ne. U tohoto typu přenosu není ztráta některého z nich nijak významným problémem. Při komunikaci mezi dvěma klienty tak spolu tito mohou bez problému simultánně mluvit.



Obr. 6.1.: Použití vstupní linky mikrofonu a následné poslání v paketech ven

6.2 Příjem zvuku a zpracování pro audio výstup

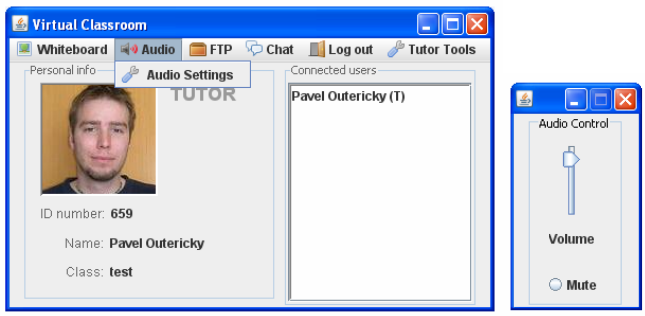
Při přijímání audio dat je postup stejný jako při kódování mikrofonního vstupu do bytového toku, jen se otočí zpracování. Nezískáme pomocí knihovny AudioSystem vstupní linku, nýbrž výstupní. A do ní se pak budou pomocí streamu data po bytech zapisovat a přehrávat na výstup v daném formátu. Tedy pokaždé, když poslouchající vlákno přijme paket s „kouskem zvuku“ ve formě bytového pole, tak ho předá výstupní lince.



Obr. 6.2: Příjem audio paketů a jejich převedení do audio streamu výstupní linky

6.3 Ovládání hlasitosti

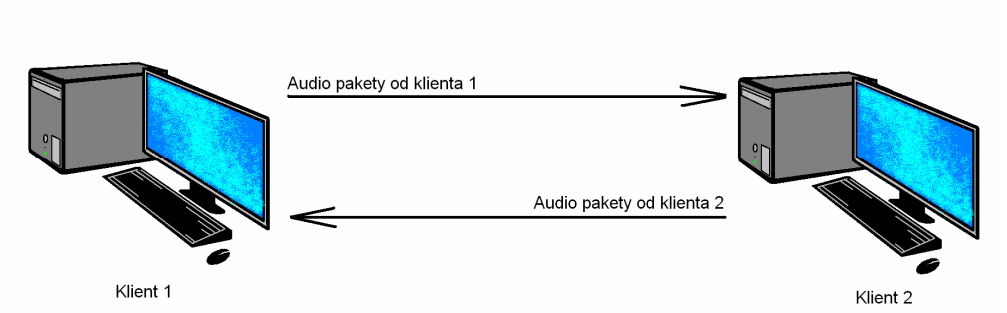
Hlasitost reprodukovaného zvuku lze ovládat pomocí kontrolního panelu pro audio, jak je zobrazeno na obr. 6.3. Lze nastavovat hlasitost pomocí jezdce nebo případně zcela ztlumit. Ve třídě, která má na starosti audio komunikaci je statická proměnná typu *float*, která určuje právě hlasitost. Tato proměnná se předává objektu typu *floatControl*, který má na starosti ovládání hlasitosti výstupní audio linky. Pokud je zaškrtnut *radiobutton* „Mute“, pak vlákno nepřijímá audio pakety, či pokud je uživatel nastaven jako mluvčí, pak nevysílá pakety. Nedělá tak vlastně nic, kromě cyklického testování *boolean* proměnné, jejíž stav mění právě onen *radiobutton*, a která určuje, zda je zvuk zapnutý či vypnutý.



Obr. 6.3: Panel pro ovládání hlasitosti a jeho položka v menu v hlavním panelu.

6.4 Řešení audio komunikace více uživatelů

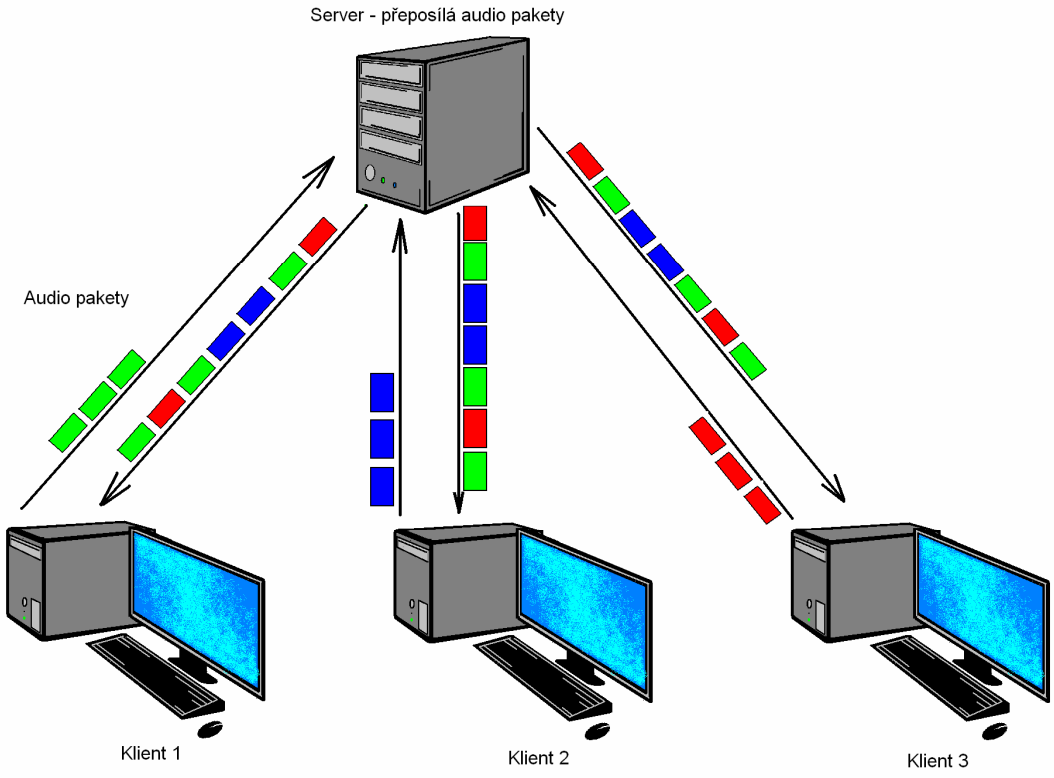
Jak jsem již zmínil, tak touto formou by spolu bez problému mohli komunikovat simultánně dva klienti. Kdy jeden posílá pakety se zvukem na adresu toho druhého na daný port (obr. 6.4).



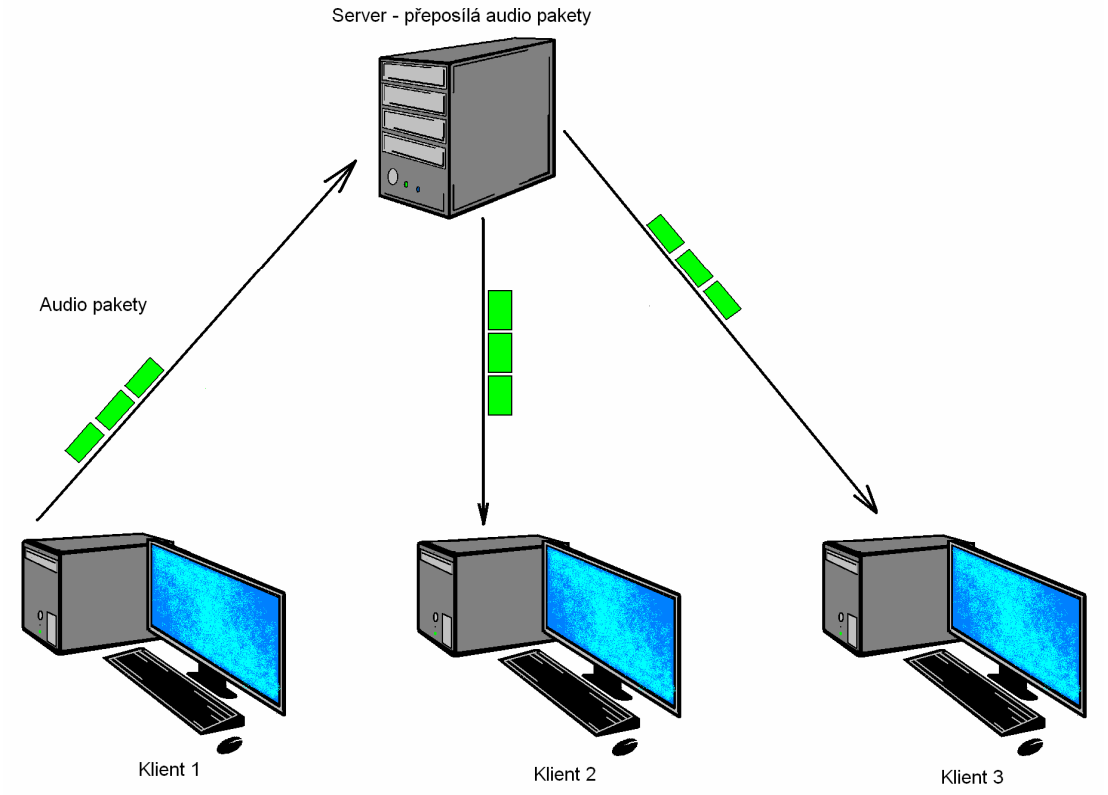
Obr. 6.4: Audio komunikace dvou klientů mezi sebou.

Ovšem v případě, že by spolu komunikovalo více klientů přes server, jak je tomu u mé práce, pak už nelze pouze posílat pakety zvuku a přeposílat je ze serveru všem klientům, jako je tomu například u paketů pro chatovací část. Došlo by samozřejmě k nežádoucímu smíchání paketů od různých uživatelů a zvuk by se stal nesrozumitelným. Tento problém by se dal vyřešit implementací nějakého audio mixeru na server. Což by ovšem byla poměrně hodně složitá záležitost a byla by nad rámec mých možností a také svým způsobem nad rámec zadání.

Já jsem tedy do svého systému implementoval audio komunikaci takovým stylem, že vždy v jednu chvíli má právo mluvit pouze jeden uživatel. Z valné většiny to bude tutor. Pokud má však například některý ze studentů nějaký dotaz či doplňující poznámku, na kterou nestačí chat, pak může tutor okamžitě přepnout uživatele, který právě může hovořit. Tedy něco jako „vyvolat“ studenta a předat mu slovo.



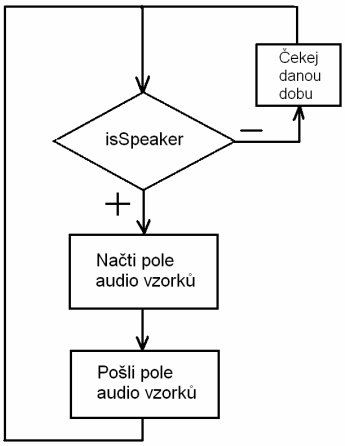
Obr. 6.5: Nežádoucí stav - smíchaní paketů při vysílání všech uživatelů současně.



Obr. 6.6.: Putování audio paketů – pouze Klient 1 je nastaven jako mluvčí, který posílá audio pakety.

6.5 Systém přepínání hovořícího klienta

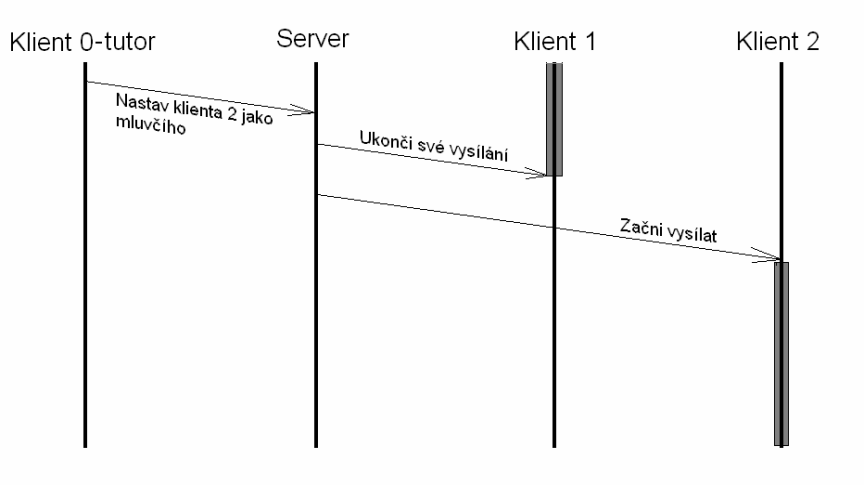
Jak bylo uvedeno výše, je audio komunikace v klientské části aplikace řešena pomocí dvou vláken. Vlákno pro příjem audio paketů a vlákno pro posílání audio paketů. Vlákno pro příjem je vytvořeno a spuštěno hned při startu klientské aplikace a úspěšném přihlášení. Druhé vlákno je také vytvořeno a spuštěno, avšak jeho běh se větví podle statické *boolean* proměnné *isSpeaker*. Pokud je tato proměnná nastavena jako pravda, znamená to, že daný klient má právě slovo a může mluvit, pak vlákno čte data z linky mikrofonu a posílá v paketech na adresu serveru. Pokud je ovšem hodnota nastavena jako nepravda, pak se vlákno vždy dostane do větve, kde pouze čeká určený časový interval a nic nedělá. Časový interval je nastaven na několik desítek milisekund. Po uplynutí intervalu opět pomocí podmínky zjistí hodnotu proměnné *isSpeaker* a podle toho se zachová. O přepínání, který klient je právě nastaven jako mluvčí a tedy má tuto proměnnou nastavenou na pravda se stará server, v kombinaci s aplikací připojeného tutora, jak bude vysvětleno níže.



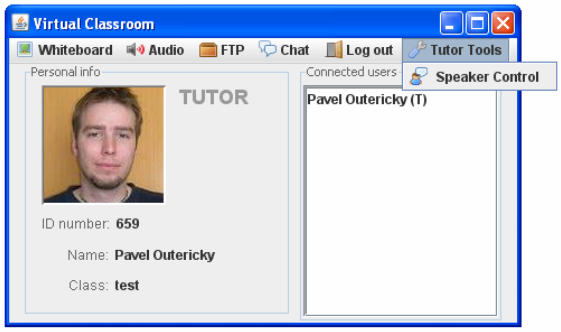
Obr. 6.7: Větvení vysílacího audio vlákna na straně klienta

Server se tedy, co se audio komunikace týče, stará krom přeposílání paketů také o nastavování právě mluvícího klienta. Pro toto se používá jeden typ řídícího paketu (viz kapitola *Řídící komunikace*), který je poslán danému uživateli, ten ho příjme a po přijetí tohoto paketu nastaví svou proměnnou *isSpeaker* na pravdu a začne tedy vysílat. Ještě předtím se však pošle ze serveru řídící paket původnímu mluvčímu, aby ukončil své vysílání. Samozřejmě pouze tehdy pokud už nějaký mluvčí zvolený je (což implicitně na začátku výuky není). Takový klient pak po přijetí tohoto paketu nastaví svou proměnnou *isSpeaker* na nepravdu. Na serveru je tedy uložena adresa právě mluvícího klienta, aby se mu mohl poslat případný „umlčovací paket“.

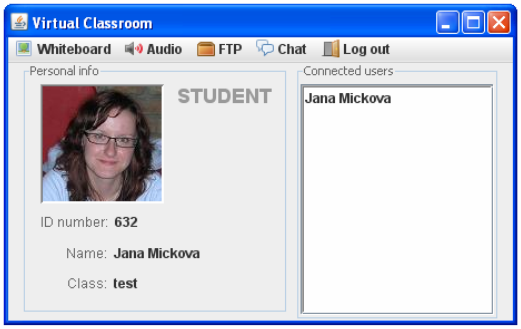
Toto přepínání iniciuje právě tutor. Pouze ten má možnost ve své aplikaci otevřít seznam pro nastavování mluvčího a pomocí kliknutí v seznamu na uživatele tak mluvčího nastavit. Tutor tedy vybere klienta pomocí daného seznamu (obr. 6.11) a tím se odešle řídící paket na server. Server pozná, že paket je pro přepínání mluvčího (viz kapitola *Řídící komunikace*), vyjme si z něj data, což je číslo (pořadí) uživatele v seznamu a zjistí jakou adresu daný uživatel má, poté pošle „umlčovací“ paket právě nastavenému mluvčímu a následně pošle paket novému mluvčímu, že může začít vysílat (obr. 6.8)



Obr. 6.8: Postup při přepínání mluvčího. Zde ukázka kdy byl aktivním mluvčím Klient 1 a tutor přikázal přepnout na Klienta 2.



Obr. 6.9: Položka menu, pomocí níž lze otevřít seznam pro volbu mluvčího. Dostupná pouze uživatelům s právy tutora.

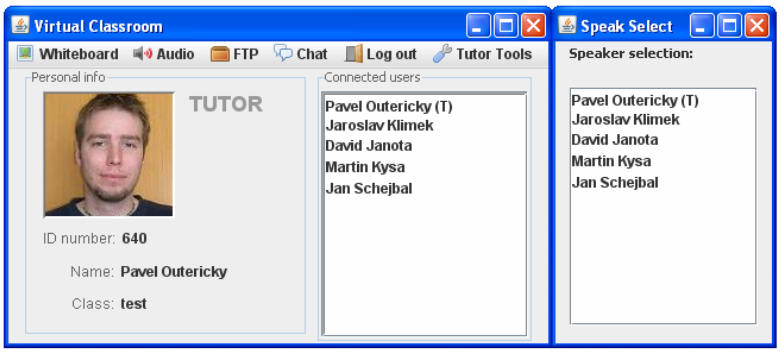


Obr. 6.10: Ukázka grafického rozhraní uživatele s právy studenta. Není dostupná položka „Tutor Tools“.

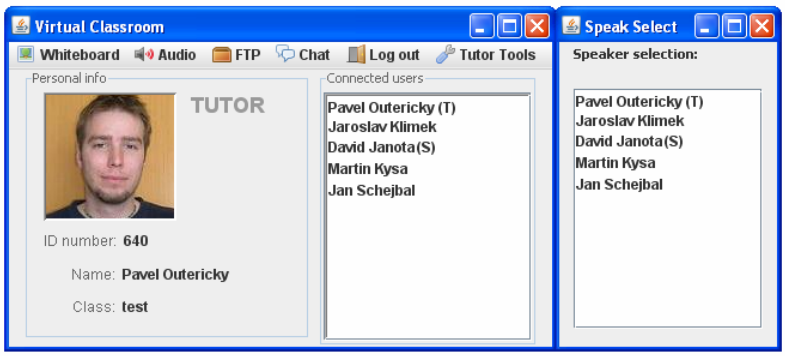
Vzhledem k tomu, že pakety jsou velmi malé (datová část paketu jsou pouhé 2 byty) a pokaždé je adresátovi jedno, kolikrát ten samý paket dostane (pokud například dostane informaci že má proměnnou změnit na *pravda*, pak je jedno, jestli to udělá jednou nebo třeba pětkrát – proměnná bude pořád *pravda*). Z toho důvodu jsem neimplementoval ověřování doručení těchto paketů, ale pouze každý z nich je posílán několikrát (zvolil jsem pětkrát), z důvodů, že by se daný paket ztratil. Na serveru je ještě jeden krok pro vyvarování se stavu, že by najednou byli aktivní dva mluvčí a tedy byl zvuk narušen. A to ten, že je pro přeposílání audio paketů nastaven filtr, který pouští dál vždy pouze pakety od jedné nastavené IP adresy.

6.5.1 Označení právě mluvícího klienta v seznamu uživatelů

Jak lze vidět z předchozích ilustrací, je právě nastavený mluvčí vždy zvýrazněn. U jeho jména je ještě přiřazeno (S) jako „Speaker“. To se přiřadí k jeho jménu vždy při tvorbě obsahu řídícího paketu se seznamem uživatelů na serveru (viz kapitola *Řídící komunikace*). Výsledek viděný u každého klienta tedy vypadá tak, jak lze vidět na obr. 6.12.



Obr. 6.11: Seznam pro volbu mluvčího. Zatím žádný uživatel není nastaven.



Obr. 6.12: Seznam pro volbu mluvčího. Uživatel David Janota nastaven jako mluvčí.

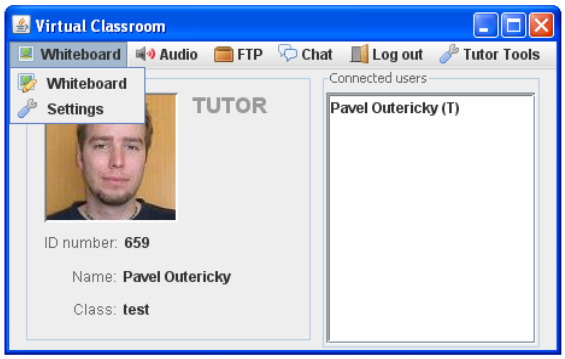
7 Whiteboard

V mé aplikaci je whiteboard nástroj několika funkcí, které poskytují dle mého názoru poměrně silnou podporu pro vzdálenou výuku a konzultaci. V této kapitole popíši postupně všechny funkce, které tento nástroj podporuje, jak z pohledu uživatelského, tak z pohledu technického, jako je například implementace v kódu a síťová komunikace.

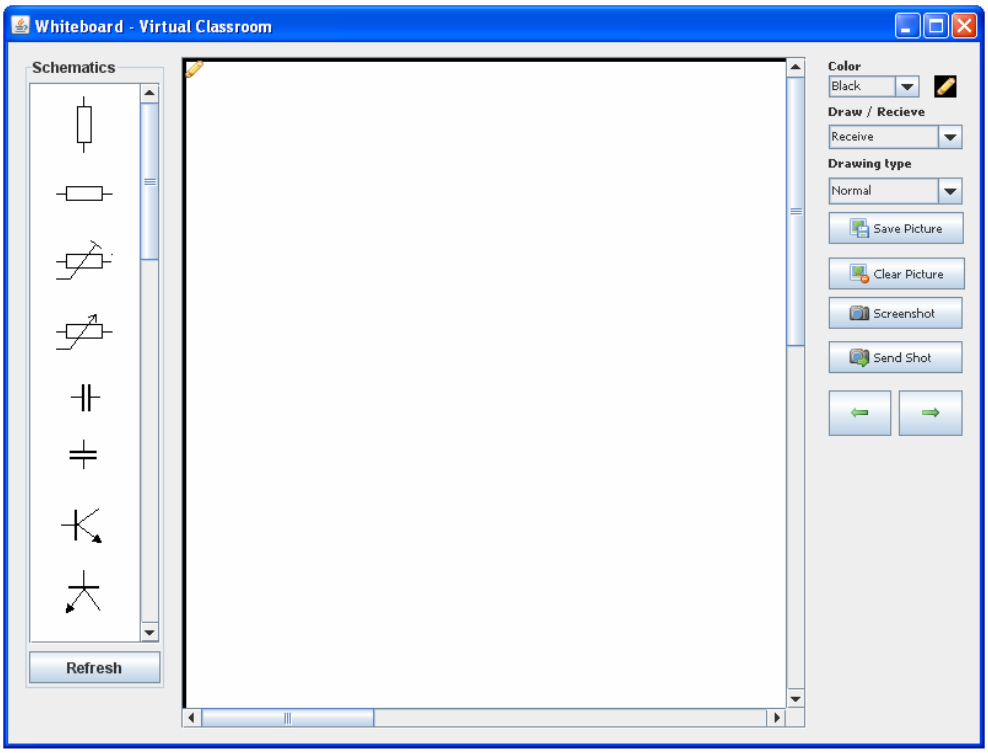
Poznámka:

Vzhledem k tomu, že při většině funkcí u nástroje whiteboard slouží server pouze k přeposílání paketů všem ostatní uživatelům, budou některé ilustrace, u nichž je to vhodné a zjednodušující, znázorněny pouze z pohledu dvou klientů a server bude z ilustrace vynechán i přes to, že pakety putují přes něj. Ne, že by tedy mezi sebou komunikovali pouze tito dva klienti, jen je v ilustraci důležité znázornit pouze dva koncové body a ne server a ostatní klienty.

Po spuštění nástroje whiteboard se před uživatelem otevře okno tohoto nástroje (obr. 7.2). Grafické komponenty a ovládací prvky v okně jsem se snažil uspořádat tak, aby výsledný celek byl co nejvíce uživatelsky přívětivý. Dominantní je kreslící a prezentační plocha, která zabírá většinu okna. Po levé straně jsou zobrazeny grafické schematické komponenty určené ke vkládání na prezentační plochu. V pravé části jsou pak ovládací prvky, pomocí nichž může uživatel vyvolat požadovanou funkci, či si nastavit požadované možnosti.



Obr. 7.1: Položka whiteboard v hlavním menu.



Obr. 7.2: Okno nástroje Whiteboard – rozměr po spuštění.

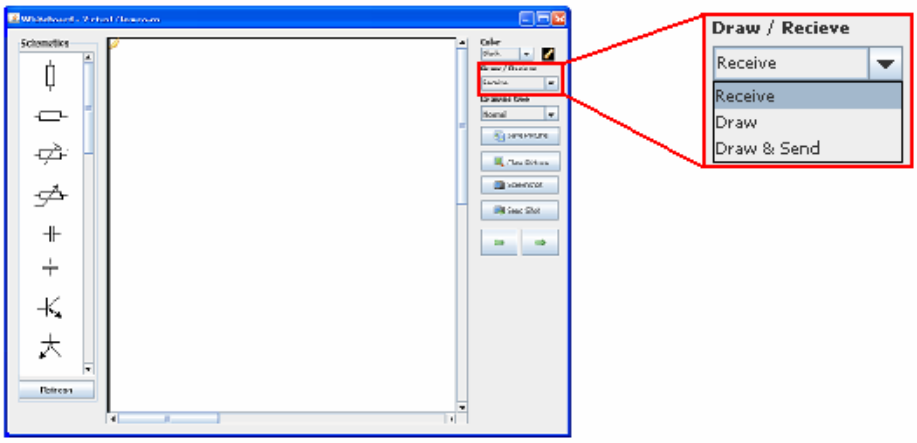
7.1 Prezentační plocha

Tato plocha je tvořena objektem typu *JPanel*, který je vložen do objektu typu *JscrollPane*, který umožňuje scrollování v případě, že není vidět celá prezentační plocha. Je to také jediný objekt v okně whiteboard, který reaguje svou velikostí na změnu velikosti okna (obr. 7.2 a 7.3). Plocha je implicitně nastavena na velikost 1600 krát 1200 pixelů. Princip plochy je v tom, že se na ni vykresluje obrázek typu BufferedImage. Při každé změně tohoto obrázku se plocha tímto obrázkem znova překreslí. Tedy pokud například při používání kreslení nakreslím čáru, tak se čára nakreslí nejprve do onoho objektu BufferedImage a ten se pak vykreslí na plochu.

Obr. 7.3: Okno nástroje Whiteboard – roztažení na celou obrazovku.

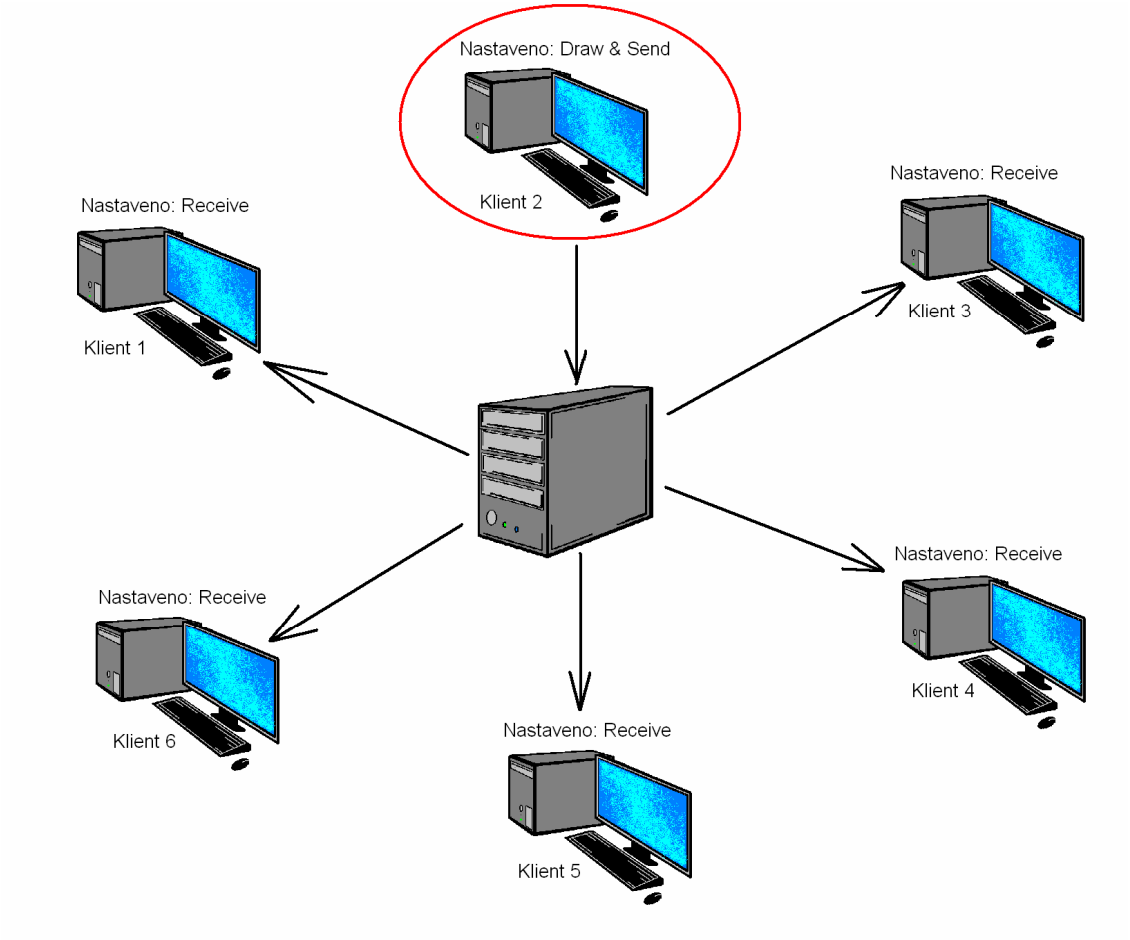
7.2 Ovládání whiteboard

Každý uživatel, ať tutor či student, má k dispozici jeden *combobox* (obr. 7.4), ve kterém si zvolí zda chce pouze přijímat, tedy zkrátka bude v pozici pasivního diváka co se prezentace týče, nebo zda chce aktivně ovládat prezentaci a ostatní výukové prvky a prezentovat tak ostatním (což je většinou tedy funkce tutora). Nebo zvolí možnost, kdy sice může ovládat všechny podporovaná prezentační prvky, avšak bez komunikace s ostatními uživateli. Tedy pouze pro vlastní potřebu (pokud například chce něco nakreslit, nebo sestavit ze schematických značek a pak například odevzdat jako výsledek tutorovi, tak nechce aby to viděli ostatní).



Obr. 7.4: Combobox pro volbu typu ovládání whiteboard.

Co se implementace v kódu týče, tak každá reakce na uživatelem vykonanou akci (ať už je to pohyb myší či kliknutí na tlačítko pro přesun na další stránku v prezentaci), se větví dle právě zvolené položky v tomto *comboboxu*. Tedy pokud student má nastaveno například, že je pouze pasivní příjemce a bude chtít kreslit či přesunout prezentaci na další stránku, tak to nebude možné, nebude se dít nic. A naopak. Implicitně při startu je v kódu nastaveno pasivní přijímání. Tedy na začátku stačí, aby se pouze tutor přepnul do stavu, kdy bude moci vysílat a prezentovat, protože všichni ostatní jsou automaticky nastaveni do pasivního přijímacího módu (obr. 7.5).



Obr. 7.5: Příklad nastavení ovládání whiteboard – Pouze klient 2 posílá pakety, protože ovládá prezentaci a ostatní pouze přijímají a sledují.

7.3 Prezentace připravených slajdů

Zde v nástroji whiteboard je možné mj. prezentovat dříve připravené grafické materiály ve formě obrazových souborů jako je například jpeg. Tutor si tedy musí před výukou připravit tyto materiály, například si pomocí funkce screenshot (viz níže) „nafotit“ prezentaci v PowerPointu. Ve chvíli, kdy je prezentace připravena, je zapotřebí ji předat dál studentům, aby si ji mohli nahrát do své aplikace a byli tak připraveni na hodinu. Tohle lze v mé aplikaci udělat pomocí FTP klienta (viz kapitola *Sdílení souborů*), či použít úplně jiné metody, jakou jsou například e-learningové systémy pro ukládání studijních materiálů nebo například uložit na jakékoliv stránky, odkud si to budou moci studenti stáhnout. Pro správnou funkci pak stač, aby si studenti i tutor nahráli před danou hodinou tyto prezentační obrázky do daného adresáře na svém počítači.

Implicitně jsem tento adresář nastavil na „GLIB/PREZENTACE“ v hlavním adresáři, odkud je spouštěna aplikace virtuální třídy. Nyní je prezentace připravena a výuka může začít.

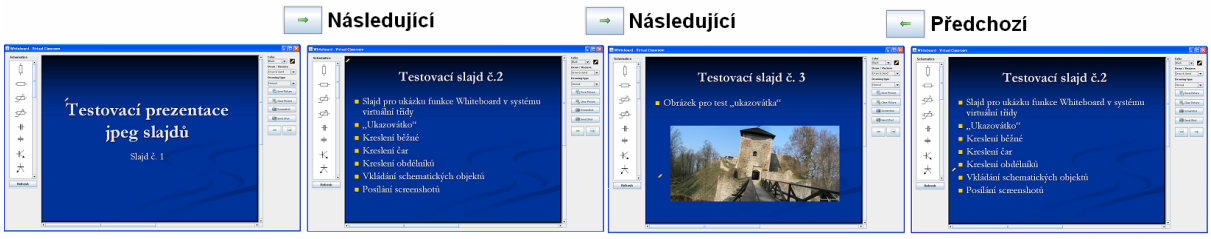
Při spuštění okna whiteboard si program automaticky vytvoří objekt typu *File*, který obsahuje adresář s prezentačními slajdy. Tento adresář si sám seřadí, protože na různých operačních systémech mohou být soubory jinak řazeny. K seřazení je použita knihovna commons (Dostupná na: http://commons.apache.org/io/?ref=darwinports.com).

7.3.1 Listování mezi slajdy

V ovládací části jsou dvě tlačítka s šipkami doleva a doprava, která intuitivně slouží k ovládání prezentace. Nyní, hned po startu nástroje whiteboard, pokud jsem jako tutor nastavený na aktivní ovládání prezentace a zmáčknu kterékoli z těchto dvou tlačítek, objeví se mi (a ostatním uživatelům) první slajd. Uvedu v krocích postup reakce na první zmáčknutí tlačítka.

1) Program načte první soubor ze seřazeného adresáře jpeg souborů.
2) Tento soubor si převede do proměnné formátu *BufferedImage.*
3) Vyčistí celou prezentační plochu (vykreslí ji celou bílou barvou).
4) Vykreslí tento BufferedImage na prezentační plochu.
5) Pošle na server informaci o tom, že se má zobrazit první slajd (server to přepošle ostatním uživatelům)

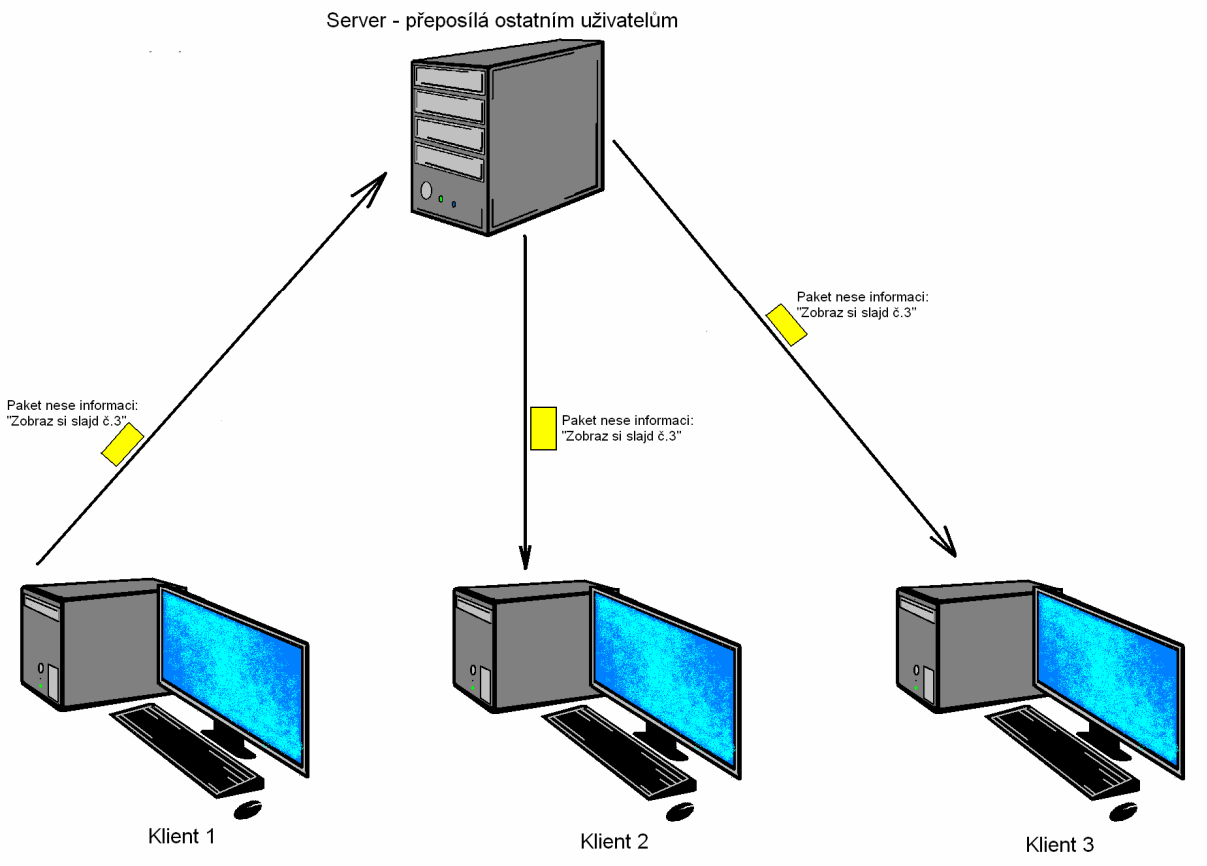
Pokud už tedy je první slajd zobrazen, tak tlačítka mají funkci posouvání na další slajd vpřed nebo vzad. V kódu je ve třídě, která se o tuto režii stará, statická proměnná, která uchovává pořadí zobrazovaného slajdu. Tedy při každém přepnutí na další nebo předchozí slajd se zopakují výše popsané kroky, akorát se nenačte první soubor, ale soubor, který je na řadě. Tedy lze posouvat slajdy stejně jako například při prezentaci v PowerPointu.



Obr.7.6: Ukázka přepínání slajdů

7.3.2 Síťová komunikace při listování mezi slajdy

Jak jsem zmínil, tak při posunu na další či předchozí slajd je poslána informace o tomto posunu na server. Paket pro informaci o zobrazení dalšího slajdu tedy nese dvě informace. Informaci o tom, jaký je to paket (whiteboard používá více typů paketů směrovaných na jeden port-popsáno níže), a číslo stránky, která se má zobrazit. Server tento paket přepošle všem klientům a klient, když ho přijme, zjistí, že je to paket pro přesun na nějakou stránku, získá z něj číslo stránky a zobrazí si ji (viz postup výše a obr. 7.7).



Obr. 7.7: Schéma síťové komunikace při přechodu na další slajd. Počáteční stav je, že všichni mají zobrazen například slajd č. 2. a schéma ukazuje co se děje poté, co tutor přepne na následující slajd.

7.4 Ukazovátko

Do nástroje whiteboard je implementováno také tzv. ukazovátko. Jedná se v principu o grafický objekt typu *JLabel*, na nějž je zobrazen obrázek malé tužky s průhledným okolím (obr. 7.8). Pokud je uživatel nastaven, že aktivně ovládá prezentaci, pak při jeho pohybu myší nad prezentační plochou je kurzor sledován právě touto grafickou komponentou v podobě malé tužky. K tomuto účelu je využito rozhraní *MouseMotionListener*. Při najetí kurzoru nad prezentační plochu se začnou sledovat souřadnice kurzoru a podle nich se bude také měnit poloha zmíněného ukazovátka jakožto grafické komponenty a při každé změně se pošle paket se souřadnicemi kurzoru na server a přepošle se všem ostatním. Ti z něj získají souřadnice a podle nich nastaví své ukazovátko. Výsledný efekt je, že ostatní uživatele vidí živě tutorův pohyb myší nad prezentační plochou a tutor tedy muže cokoliv tímto způsobem ukazovat (ukazovátko v ploše lze vidět např. na obr. 7.11).

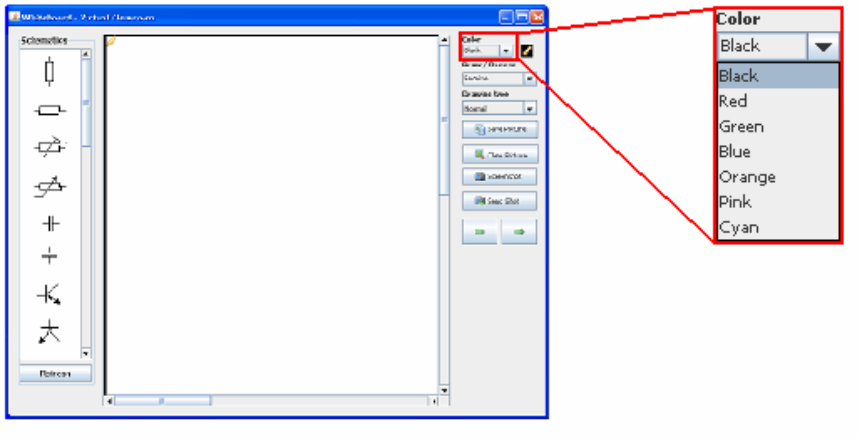
Obr. 7.8: Ukazovátko

7.5 Kreslení

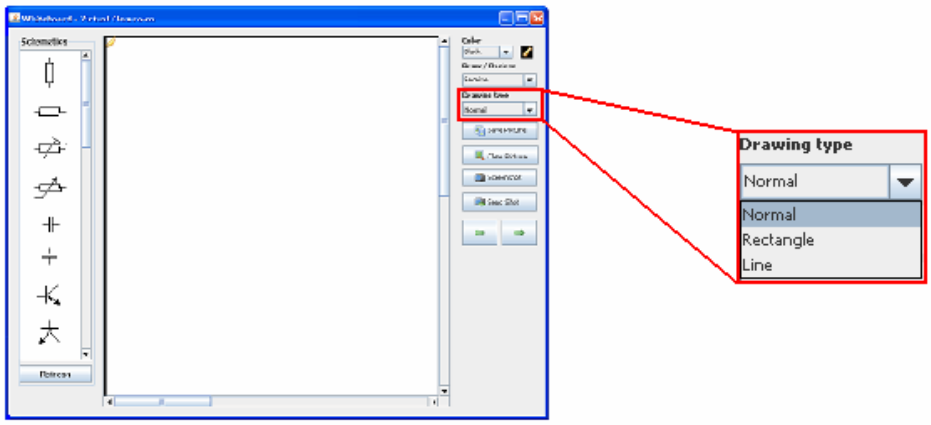
Na prázdnou prezentační plochu, či přímo do zobrazovaných slajdů lze také kreslit. Kreslit samozřejmě také může pouze uživatel, který je nastaven jako aktivní pro ovládání prezentace. Kreslení je implementováno způsobem, kdy je opět využito rozhraní *MouseMotionListener* spolu s rozhraním *MouseListener*, spjaté s komponentou prezentační plochy. Tedy ve chvíli, kdy uživatel stiskne levé tlačítko myši nad prezentační plochou, aktivuje tak touto událostí funkci, která sleduje pohyb myši a podle tohoto pohybu vykresluje na prezentační plochu trasu myši tak dlouho, dokud není tlačítko uvolněno.

7.5.1 Barvy a tvary

Při kreslení je možné také zvolit jednu ze základních barev. Pro volbu barvy slouží k tomu určený *combobox* (obr. 7.9). Dále jsou také tři typy kreslení, kdy uživatel může zvolit normální typ, při kterém kreslí tak, jak se pohybuje myš, nebo kreslit rovné čáry z jednoho bodu do druhého a nebo obdélníky, kdy se určí levý horní roh a pravý dolní roh. Pro tuto volbu je také použit *combobox* (obr. 7.10).



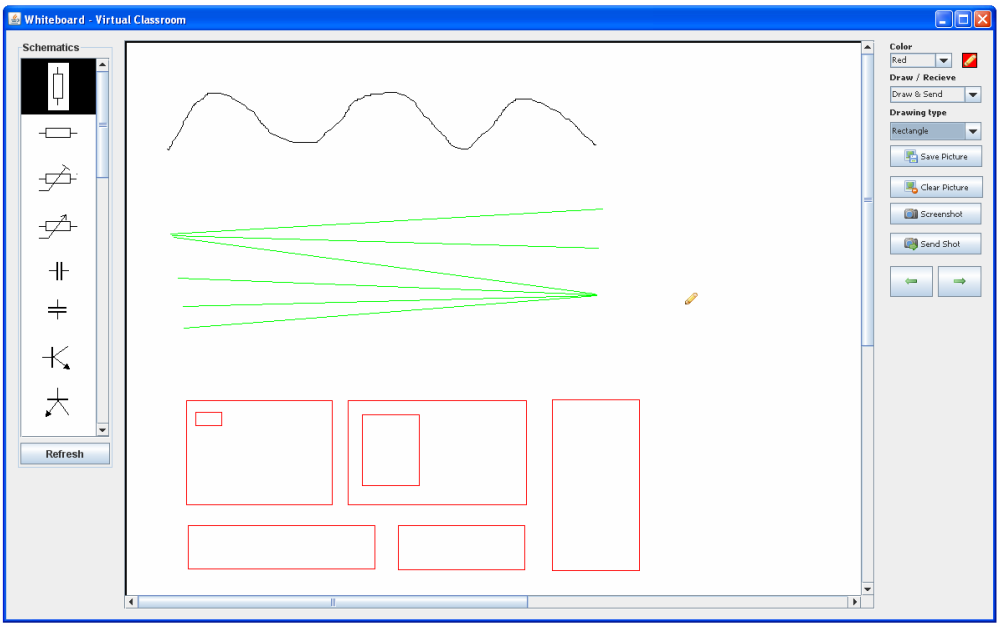
Obr. 7.9: Combobox pro volbu barvy.



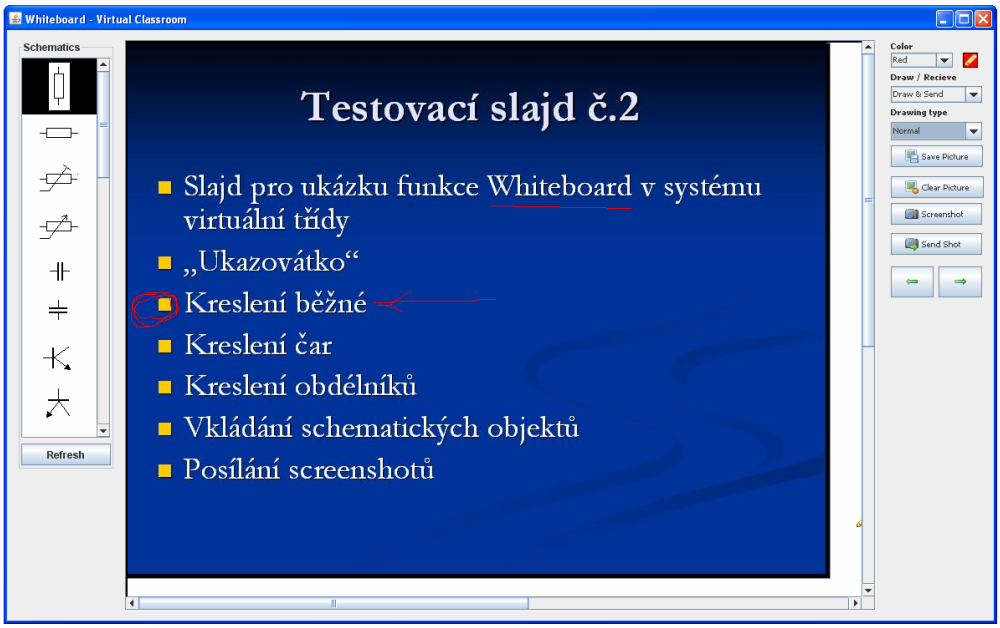
Obr. 7.10: Combobox pro volbu typu kreslení.

Posloupnost prováděných instrukcí při kreslení:

1) Zmáčknuto tlačítko myši nad prezentační plochou.
2) Zjištění počátečních souřadnic myši.
3) Zjištění typu kreslení podle hodnoty *comboboxu* (normální, čára, obdélníka).
4) Zjištění barvy podle hodnoty daného *comboboxu*.
5) Vykreslení, které se dělí dle typu. Pokud je normální kreslení, pak se při každém sebemenším pohybu vykreslí čára a hned se vezme konec této malé čáry jako začátek nové a výsledný efekt je tedy plynulé vykreslení stopy kurzoru. Pokud je nastavena čára, pak se čeká na uvolnění tlačítka a tam kde bylo tlačítko uvolněno se nastaví konec čáry a vykreslí se linka mezi těmito body. V případě obdélníku je princip stejný jako u čáry, avšak se mezi těmi dvěma body vykreslí obdélník. Vše v takové barvě jaká je právě nastavena.



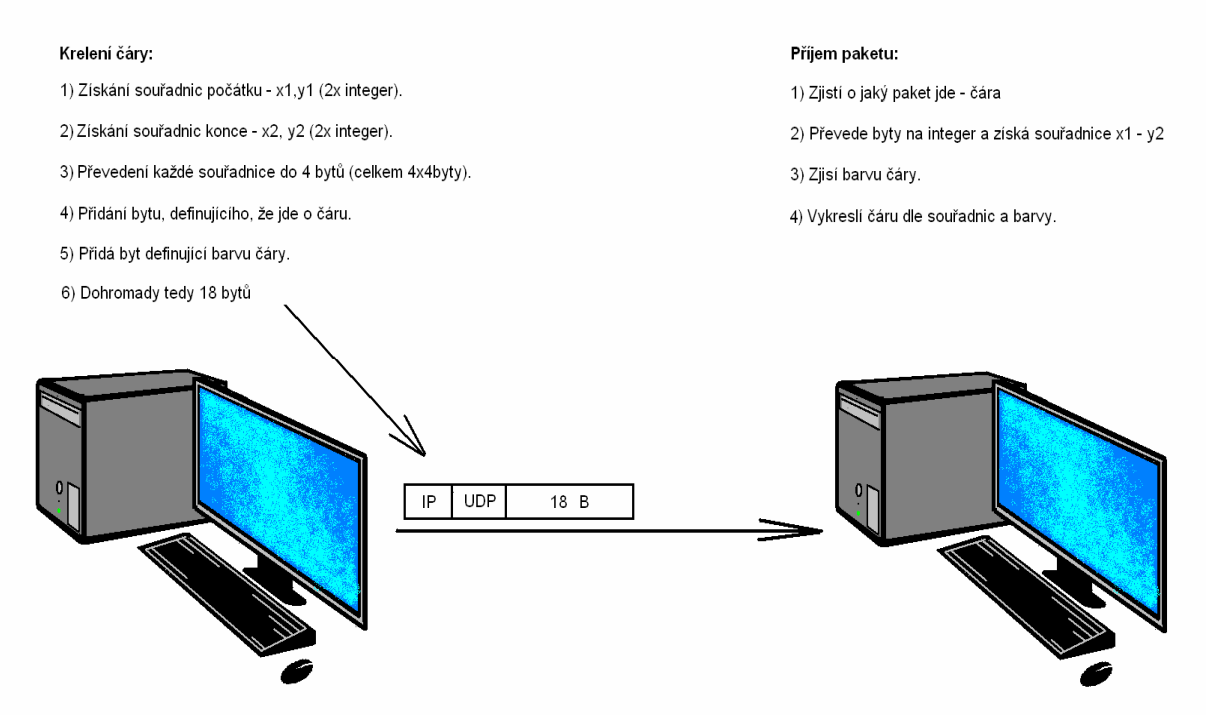
Obr. 7.11: Ukázka kreslení. Nahoře nastavena černá barva a normální kreslení, kdy je kopírován pohyb myši, níže rovné čáry a dole ukázka kreslení obdélníků.V pravé části lze vidět ukazovátko.



Obr. 7.12: Ukázka kreslení i s prezentačním slajdem.

7.5.2 Síťová komunikace při kreslení

Při kreslení probíhá síťová komunikace zhruba stejně jako u ukazovátka. Princip je takový, že při každé změně či akci se uloží počáteční souřadnice myši a koncové souřadnice myši. Tedy při normálním kreslení se neustále při každém pohybu inovují tyto souřadnice a posílají. Při kreslení čáry se vezme její začátek a a konec a při obdélníku jeho levý horní a dolní pravý roh. Tyto souřadnice se uloží do paketu spolu s identifikací, jestli se jedná o normální kreslení, čáru nebo obdélník a ještě s určením barvy. Tedy je zapotřebí šest hodnot: čtyři souřadnice, barva a typ. Souřadnice jsou ve formátu integer, který se pak pomocí bitových posunů převádí do pole bytů a typ kreslení je jeden byte.

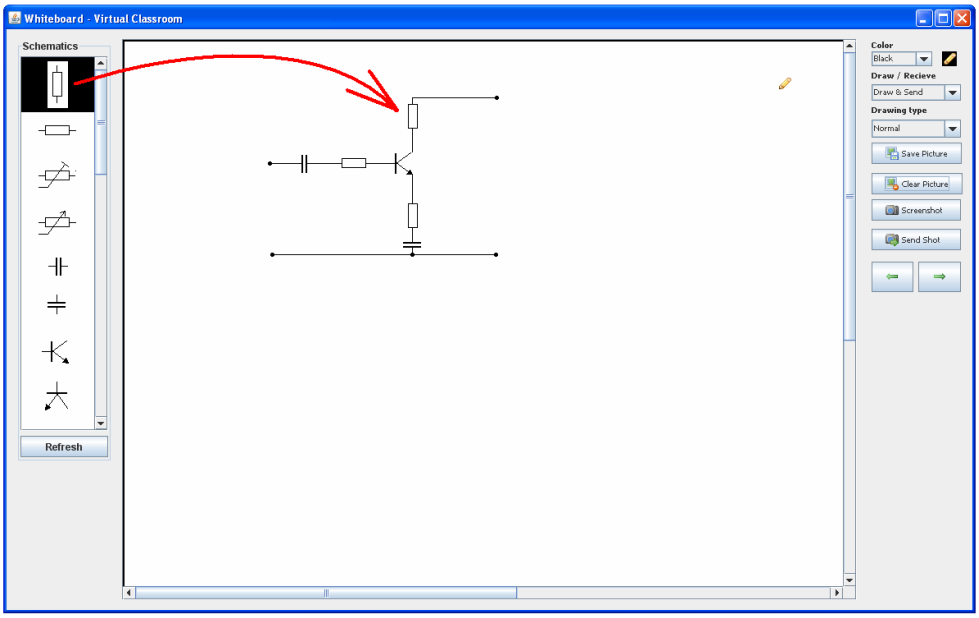


Obr. 7.13: Princip přenášení dat o kreslení (ukázka přenosu čáry).

7.6 Vkládání grafických objektů

Další užitečnou funkcí, která je ve whiteboard implementována, je možnost vkládání grafických objektů, tedy například schematických značek. Před spuštěním programu lze do daného adresáře, který je implicitně nastaven v kódu na /GLIB nahrát grafické objekty v podobě grafických souborů, například typu jpeg nebo spíše png. Po spuštění programu a nástroje whiteboard je pak možné tyto značky pomocí funkce drag & drop vkládat na prezentační plochu. Vysvětlím na příkladu. Já jsem si nakreslil dvaadvacet ukázkových elektrotechnických schematických značek. Každá z těchto značek je uložena jako obrazový soubor. Velikost těchto obrázků se pohybuje kolem 50 krát 50 pixelů a jsou uloženy ve formátu png. Stejně jako to bylo uvedeno u prezentačních slajdů, tak i zde si program při spuštění nástroje whiteboard automaticky načte tento adresář a seřadí abecedně soubory v něm. Tyto soubory, tedy obrázky, pak zobrazí do *JList* komponenty. Nad touto komponentou a nad prezentační plochou je použito rozhraní pro práci s drag & drop funkcí, tedy pro přetažení objektu myší. Pokud uživatel tedy „uchopí“ myší některý obrázek z daného *JListu* a přetáhne ho na prezentační plochu. kde ho „upustí“, tak při upuštění jsou získány souřadnice a na tyto souřadnice je daný objekt vykreslen (obr. 7.14). Stejně jako u prezentačních slajdů je i zde nutné, aby všichni uživatele měli naimportované (tedy nahrané v daném adresáři) stejné schematické značky. Vykreslovaná značka není totiž určena jejím jménem, ale jejím pořadím v seřazeném seznamu. Kroky při vykreslení:

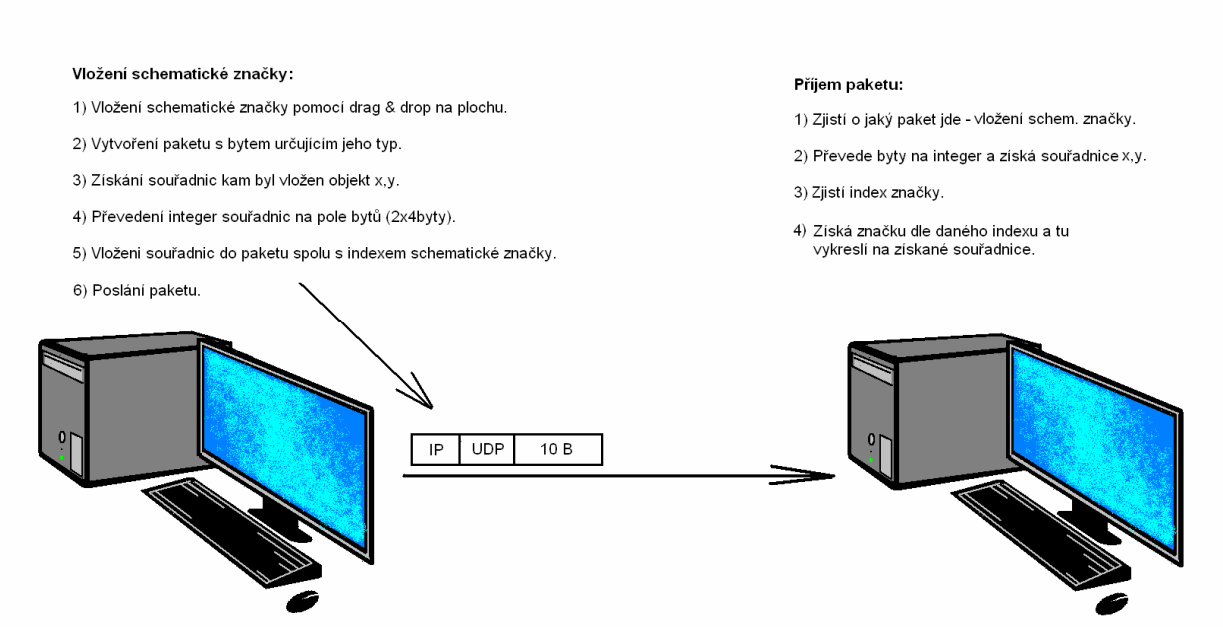
1) Uchopení značky z daného seznamu (řešeno pomocí implementace drag & drop funkce) a její tažení na prezentační plochu (na této ploše se kurzor změní ve značku, která informuje, že je možno objekt vložit)
2) Upuštění objektu a získání souřadnic kurzoru.
3) Získání grafického objektu ze souboru v adresáři podle pořadí.
4) Vykreslení tohoto grafického objektu na dané souřadnice.
5) Poslání souřadnic s indexem objektu v paketu směrem k serveru.



Obr. 7.14: Ukázka tvorby pomocí vkládání schematických grafických objektů.

7.6.1 Síťová komunikace při vkládání grafických objektů

Po vložení a vykreslení dané značky se její souřadnice a její pořadí pošle v paketu na server a ke klientům se tedy dostane paket s informací, že si mají vykreslit n-tou značku ze seznamu na souřadnice x,y.



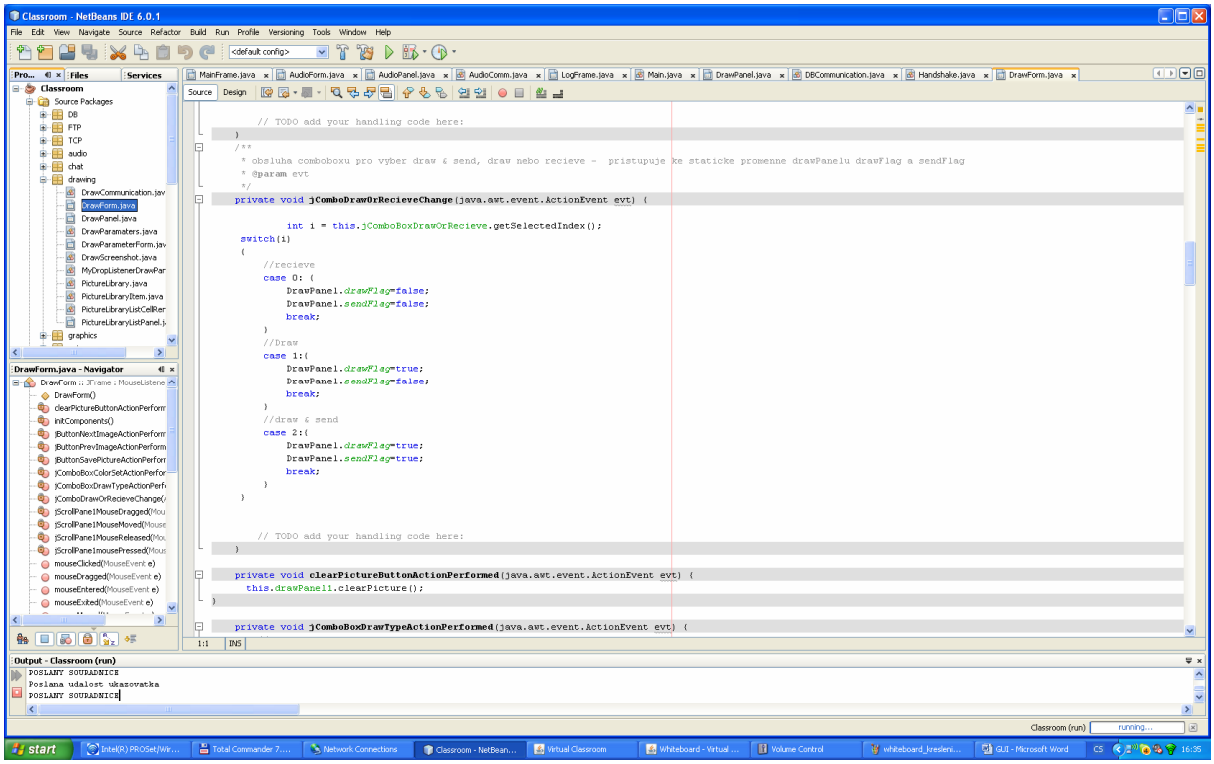
Obr. 7.15: Schéma komunikace při vložení grafického objektu.

7.7 Screenshot

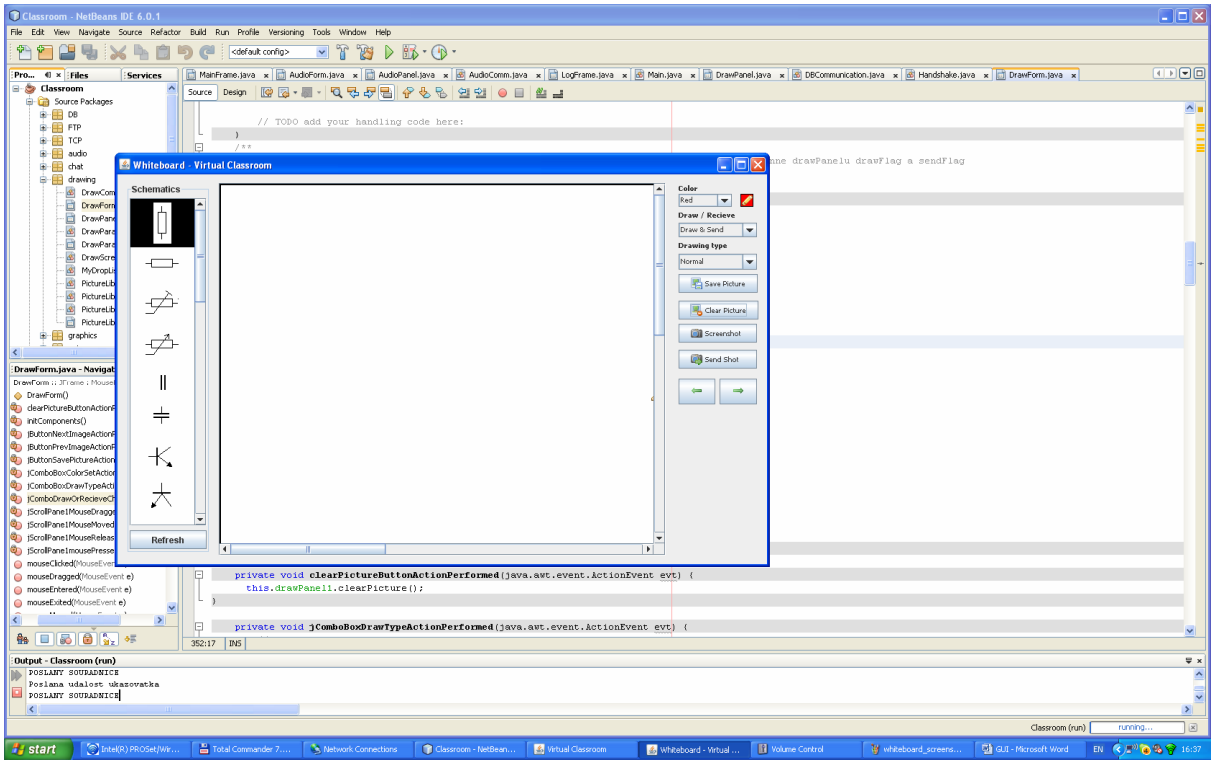
Další funkcí nástroje whiteboard je možnost sejmutí části vlastní obrazovky a poslání ostatním uživatelům. Pro snímání obrazovky je použita knihovna *Java.awt.Robot.* Pomocí níž se dá sejmout obrazovka či část obrazovky určená obdélníkem a vložit přímo do objektu typu *BufferedImage.* Určení části obrazovky, která má být vyfocena se provádí pomocí velikosti zobrazované prezentační plochy. Zmenšením nebo zvětšováním okna whiteboard se nám zvětšuje nebo zmenšuje také prezentační plocha, nebo spíše přesněji řečeno *ScrollPane*, ve kterém je plocha umístěna. Pro vytvoření screenshotu obrazovky se tedy jako počátek použije horní levý roh onoho *ScrollPane* a jako konec se požije jeho dolní pravý roh. Poté, po zmáčknutí tlačítka „Screenshot“, se vše co je na obrazovce za touto komponentou „vyfotí“, uloží v paměti do objektu typu *BufferedImage* a zobrazí se na vyčištěnou prezentační plochu (obr. 7.16 až 7.19).

Postup při „vyfocení“ části obrazovky:

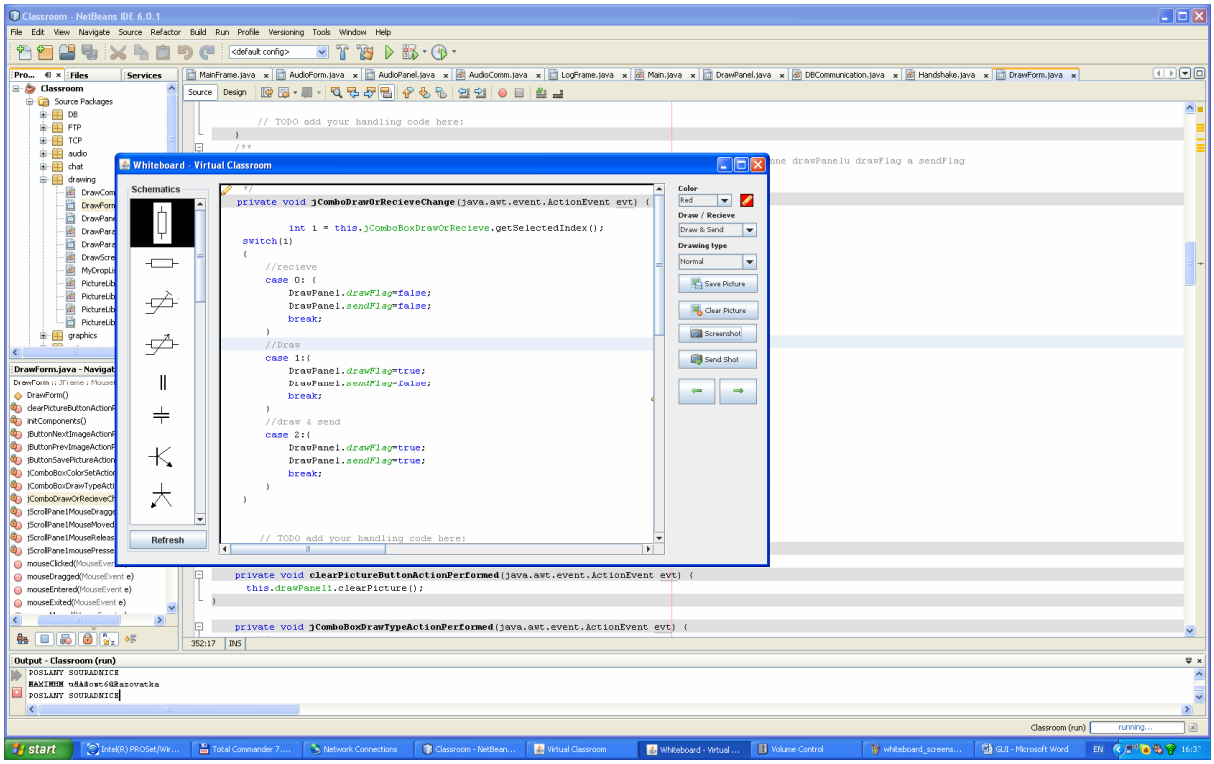
1) Přes zobrazovanou část, kterou chci vyfotit překryji okno nástroje whiteboard.
2) Nastavím velikost podle toho, co přesně chci vyfotit (obr. 7.17).
3) Zmáčknu tlačítko „Screenshot“. V tuto chvíli program získá polohu daného *ScrollPane* na obrazovce a tu uloží jako souřadnice počátku screenshotu, a přičte k souřadnicím velikost tohoto *ScrollPane* a tím získá koncový bod screenshotu.
4) Okno nástroje whiteboard se nastaví jako neviditelné (aby se vyfotilo to co je za ním).
5) Pomocí zmíněné knihovny se vyfotí daná část obrazovky (sejmutí obrazovky je empiricky nastaveno tak, že se provádí pětkrát z důvodu, že při jednom sejmutí se nemusí sejmout všechny překrývají se okna na obrazovce a screenshot by tedy nebyl úplný) a uloží se do objektu *BufferedImage*.
6) Vyčistí se prezentační plocha.
7) Sejmutý obrázek se na plochu vykreslí (obr. 7.18).



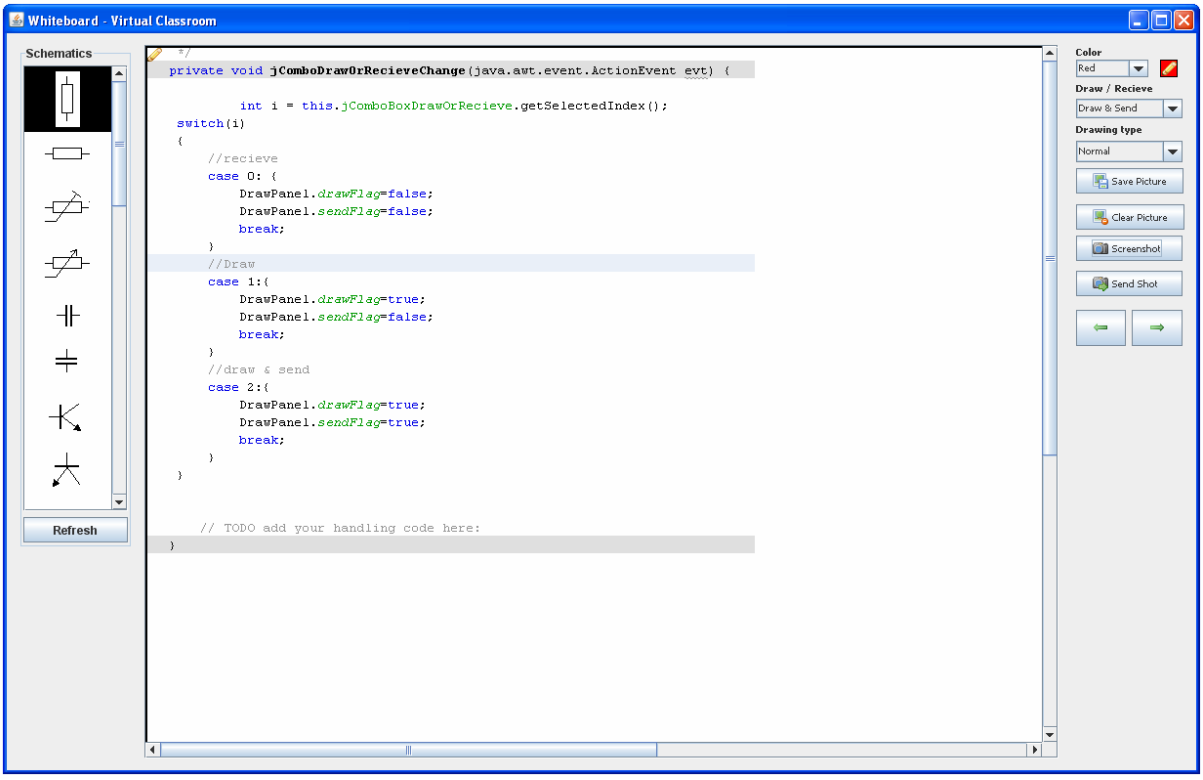
Obr. 7.16: Ukázka obrazovky (Vývojové prostředí programu NetBeans), ze které chci například vyfotit pouze kus kódu. Zde třeba kód viditelné metody.



Obr. 7.17: Nastavení okna whiteboard před sejmutím screenshotu.



Obr. 7.18: Okno whiteboard po sejmutí screenshotu.



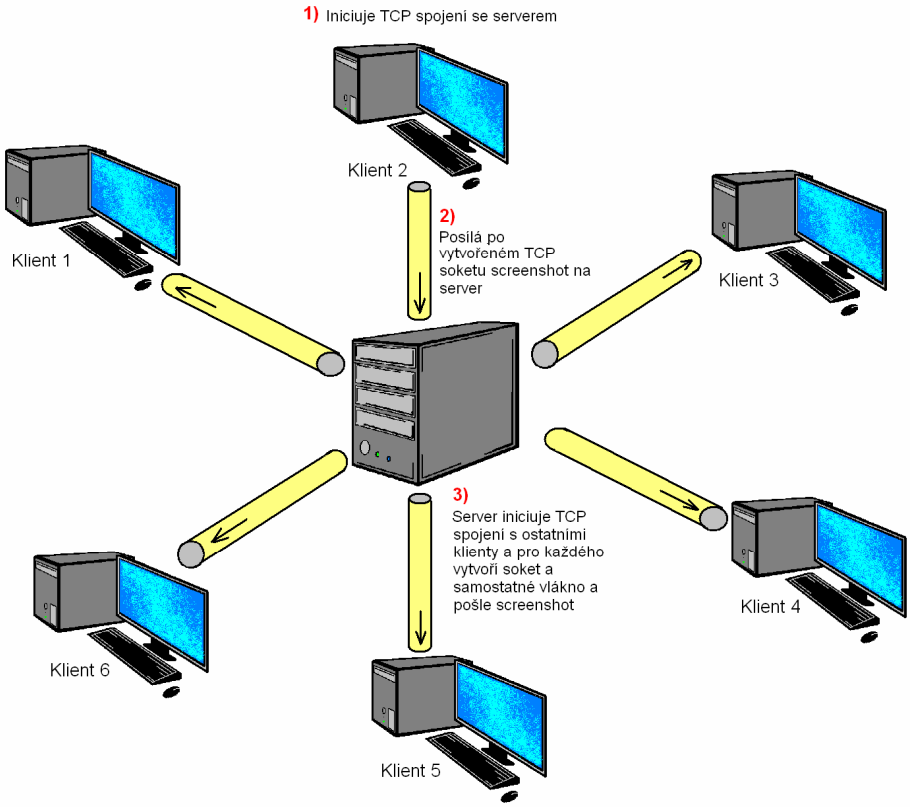
Obr. 7.19: Výsledný efekt, ukázaný pro názornost na roztaženém okně whiteboard. Takto je vidět u daného uživatele a stejně tak se zobrazí po poslání i ostatním uživatelům.

Jak lze vidět z přiložených ilustračních obrázků, tak při snímání screenshotu bohužel uživatel nevidí, co přesně sejme. Ideálním řešením by bylo částečné zprůhlednění dané komponenty a i komponenty *JFrame*, ve které se nachází. Ovšem v Java SE nejsou tyto možnosti zrovna příliš dobře a snadno řešitelné. Například platforma .NET má velice snadnou implementaci takového zprůhlednění komponenty i jejího podkladu, avšak v Java SE se mi nepodařilo najít nějaké schůdné řešení, kterým bych zprůhlednil ono okno a dodal tak aplikaci lepší uživatelskou přívětivost. Ale i tak si myslím, lze tuto funkci screenshotu velice dobře používat.

7.7.1 Přenos screenshotu

Pokud máme nějaký obrázek výše popsaným způsobem vyfocený a zobrazený, lze ho jednoduše pomocí jednoho tlačítka „Send Shot“ poslat ostatním uživatelům, kterým se okamžitě zobrazí. Zvolil jsme přenos pomocí TCP z důvodu, že se jedná o přenos celého obrázku, jehož části musí být doručeny všechny a ve správném pořadí. Nejlépe bude asi opět popsat princip této komunikace v krocích, které nastanou po zmáčknutí tlačítka „Send Shot“:

1) V paměti uložený screenshot, který zatím není komprimován se z důvodu zmenšení zkomprimuje do jpeg formátu.
2) Vytvoří se objekt pro poslání daného objemu dat pomocí TCP soketu, který se spojí s daným portem na serveru.
3) Na serveru na daném portu poslouchá (akceptuje uživatelské žádosti) objekt typu *ServerSocket*. Jakmile je přijata žádost od klienta, který na server posílá screenshot pak server vytvoří samostatný TCP soket pro tohoto klienta a přijme data od klienta posílaná. Po přijetí všech dat se soket uzavře.
4) Poté, co byl přenos celého screenshotu od zdrojového klienta na server dokončen se vytvoří pro každého klienta soket, který se s daným klientem spojí (na každé klientské aplikaci běží také vlákno, kde objekt *ServerSocket* čeká na nějakou iniciativy pro navázání TCP spojení zvenčí). Pak se paralelně obsluhují všichni klienti a každému se tak doručí onen screenshot. Po dokončení přenosu se všechny soketu uzavřou.
5) Když je klientovi screenshot úspěšně doručen, tak se vyčistí jeho prezentační plocha a screenshot se na ni vykreslí.

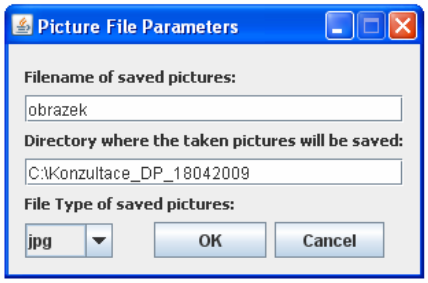


Obr. 7.20: TCP komunikace při posílání screenshotu. Klient 2 posílá screenshot ostatním uživatelům

7.8 Možnost záznamu

V mé virtuální třídě je implementována i možnost záznamu prezentované látky. A to tak, že při zmáčknutí tlačítka „Save Picture“ se celá prezentační plocha se vším, co je na ní vykreslené, uloží do souboru dle nastavení (viz níže). Uživatel si tak může naprosto kdykoliv uložit obsah prezentační plochy. Při každém uložení se vytvoří nový soubor dle nastavení, jehož název končí číslem. Toto číslo je jakési pořadové číslo. Tedy při každém dalším uloženém obrázku se číslo inkrementuje a tak tedy lze postupně ukládat další a další soubory.

Jak lze vidět na obr. 7.1, je součástí menu whiteboard také položka „Settings“. Po zvolení se otevře okno, v němž je možné nastavit, jak se budou jmenovat uložené obrazové soubory a také do jakého adresáře se tyto soubory mají ukládat a v jakém formátu. K dispozici je formát png nebo jpeg. Na obr. 7.21 lze vidět formát nastavení. Dle tohoto nastavení se první soubor uloží jako *C:\Konzultace_DP_18042009\obrazek1.jpg* další jako *C:\Konzultace_DP_18042009\obrazek2.jpg* atd.



Obr. 7.21: Ukázka nastavení ukládání prezentační plochy do obrazových souborů.

7.9 Whiteboard shrnutí

Vzhledem k tomu, že poskytuje vizuální pomůcku s poměrně velkou možností uživatelské interakce a okamžitého záznamu v podobě uložení do obrazových souborů, je whiteboard tedy asi nejvýznamnějším nástrojem mého systému virtuální třídy. Ke komunikaci využívá jeden dedikovaný UDP port, na něhož jsou posílány pakety dle implementovaného protokolu (viz Tab. 7.1) a také využívá jeden TCP port pro posílání screenshotů.

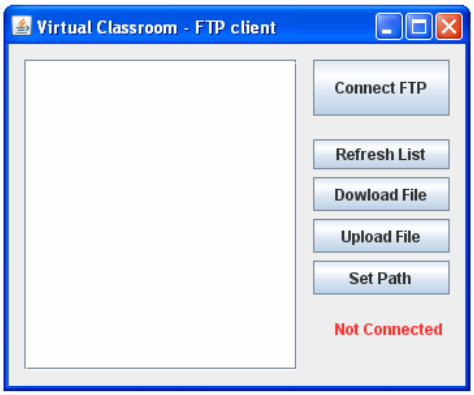
Tab. 7.1: Typy paketů pro whiteboard.

Typy paketů nástroje whiteboard	
0	Běžné kreslení
1	Kreslení obdélníku
2	Kreslení rovné čáry
3	Vložení schematické značky
4	Událost ukazovátka
5	Zobrazení daného slajdu

.

8 Sdílení souborů

Do své klientské aplikace jsem implementoval jednoduchého jednoúčelového klienta pro komunikaci s FTP serverem. Klient se snadno spustí z hlavního menu aplikace. Poté se uživateli objeví okno tohoto klienta (obr. 2.1). Nyní uživatel ještě není připojen. Pro připojení stačí zmáčknout tlačítko „Connect FTP“ a klient se automaticky připojí.

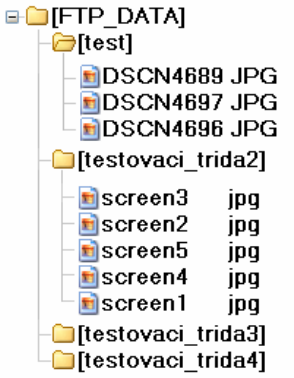


Obr. 2.1: Okno FTP klienta před připojením.

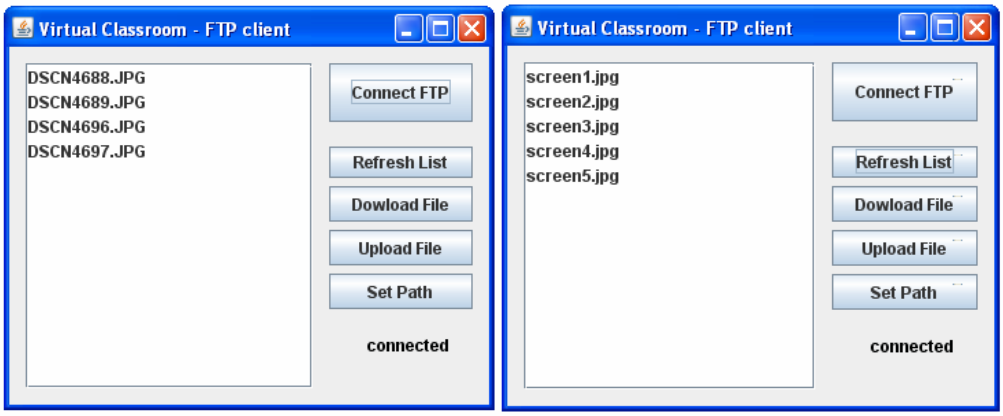
Jako adresa FTP serveru je použita adresa serveru zadaná na začátku (tedy FTP server běží na stejné IP adrese jako Classroom Server a MySQL server). Jako údaj k přihlášení se použije login klienta jako jméno klienta pro FTP server a název třídy, do které klient patří se použije jako heslo. Samozřejmě by se dalo využít jakýchkoliv údajů. Mé nastavení tedy je:

FTP login = Classroom login klienta
FTP heslo = Název třídy do které klient patří

Klientská aplikace se tedy úspěšně přihlásí na FTP server do daného adresáře (který je implicitně nastaven jako FTP_DATA). V tomto adresáři jsou další podadresáře, jejichž jména jsou stejná jako jména tříd. Uživatelova klientská aplikace se automaticky přihlásí a přesune do adresáře s názvem třídy, do které uživatel patří. Pokud tedy se já budu chtít připojit, tak se jako moje přihlašovací jméno na FTP server použije „outericky“ a jako heslo název třídy, tedy „test“ a automaticky se přesunu do adresáře FTP_DATA\test. V seznamu v okně pro FTP se okamžitě zobrazí soubory uložené na serveru v daném adresáři (obr. 8.3). A nyní mám možnost si jakýkoliv soubor stáhnout či naopak nějaký na server uložit. Dále je zde ještě možnost aktualizace obsahu adresáře. Každá třída má tedy svůj vlastní adresář na FTP serveru a při použití FTP klienta je uživatel automaticky nasměrován do tohoto adresáře. Tedy si navzájem uživatelé vyměňují soubory v rámci jedné třídy.

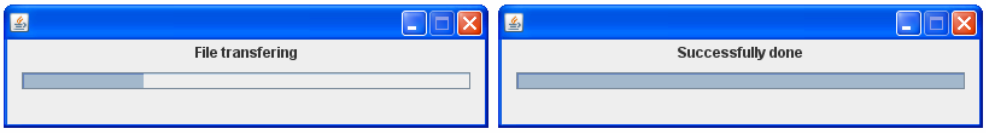


Obr. 8.2: Příklad adresářové struktury na FTP serveru. Pokud by uživatel patřil do třídy„test“, pak by se mu zobrazily soubory v adresáři „test“ nebo pokud by patřil například do třídy „testovaci_trida2“ pak by se mu zobrazily soubory v adresáři „testovaci_trida2“ (obr. 8.3)



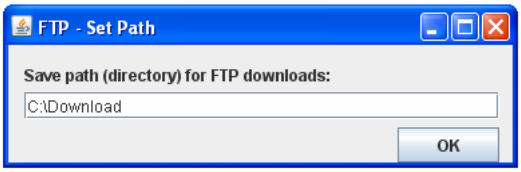
Obr. 8.3: Okno FTP klienta pro uživatele patřícího do třídy „test“ (dle Obr. 8.2) a okno FTP klienta pro uživatele patřícího do třídy „testovaci_trida2“ (dle Obr. 8.2)

K práci s protokolem FTP jsem použil volně dostupnou knihovnu *ftp4j* [Dostupná na: http://www.sauronsoftware.it/projects/ftp4j/], ze které jsem implementoval metody pro připojení k serveru a práci se soubory. Stejně tak jsem použil z této knihovny i *transfer listener* pro sledování průběhu přenosu a vytvoření progressbaru, který tento průběh ukazuje (obr. 8.4).



Obr. 8.4: Progressbar ukazující průběh přenosu a dokončení přenosu (na 1,5 sekundy se zobrazí „Successfully done“, pak progressbar zmizí.

Jak je vidět na obr. 8.3, tak je v okně také tlačítko „Set Path“. Tedy uživatel si může nastavit cestu, kam chce, aby se jím stahované soubory ukládaly.



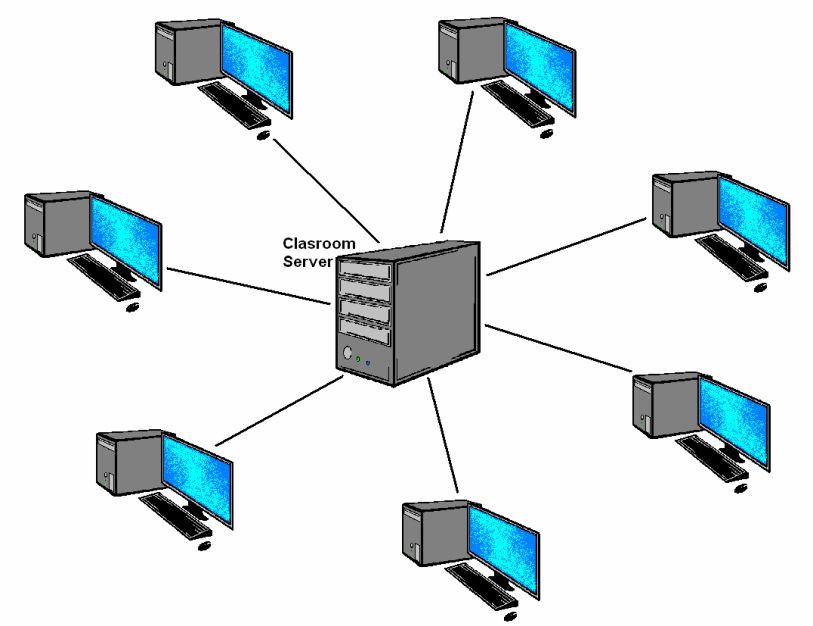
Obr. 8.5: Nastavení cesty pro ukládání souborů z FTP serveru.

9 Classroom server

Tato kapitola popisuje funkci Classroom serveru, který je v ostatních kapitolách velmi zmiňován, jakožto jakýsi most pro přeposílání paketů ostatním uživatelům. Správa klientů, kterou má tento server také mimo jiné na starosti, je poměrně podrobně popsána v kapitole *Řídící komunikace* a tedy ji zde už nebudu podruhé rozebírat. Spíše zde vysvětlím, jak je řešeno přeposílání paketů mezi ostatní klienty.

9.1 Přeposílání paketů

Valná většina komunikace, která probíhá v rámci této aplikace znamená poslat informaci od jednoho uživatele všem ostatním připojeným uživatelům (například textovou zprávu či audio paket). Původně jsem klientskou aplikaci měl vyvinutou tak, že komunikovaly dvě přímo mezi sebou. Později jsem přešel na propojení více uživatelů a vytvořil jsem tuto serverovou aplikaci, která (pokud vynecháme řídící činnost při správě uživatelů a mluvčího) se pro klienta vlastně tváří úplně stejně jako druhá klientská aplikace, se kterou komunikuje. Klient posílá pořád stejné pakety, jako když komunikoval přímo s druhou klientskou aplikací. Akorát jsou tyto pakety pak posílány více uživatelům než jen jednomu. Zkrátka, co kdo na server pošle, to přijde i všem ostatním (obr. 9.1).



Obr. 9.1: Ukázka připojení klientů ke Classroom Serveru.

Pro přeposílání paketů je využit unicast, kdy se každý paket posílá zvlášť na každou IP adresu. Princip je takový, že při přijetí paketu začne na serveru v daném vlákně cyklus, který prochází seznamem klientů (viz kapitola *Řídící komunikace*) a z každého klienta získá jeho IP adresu a na tu pošle onen paket. Samozřejmě přeskočí položky seznamu, které jsou null. Takto projde celý seznam a až dojde na konec, tak se vrátí do stavu čekání na další paket.

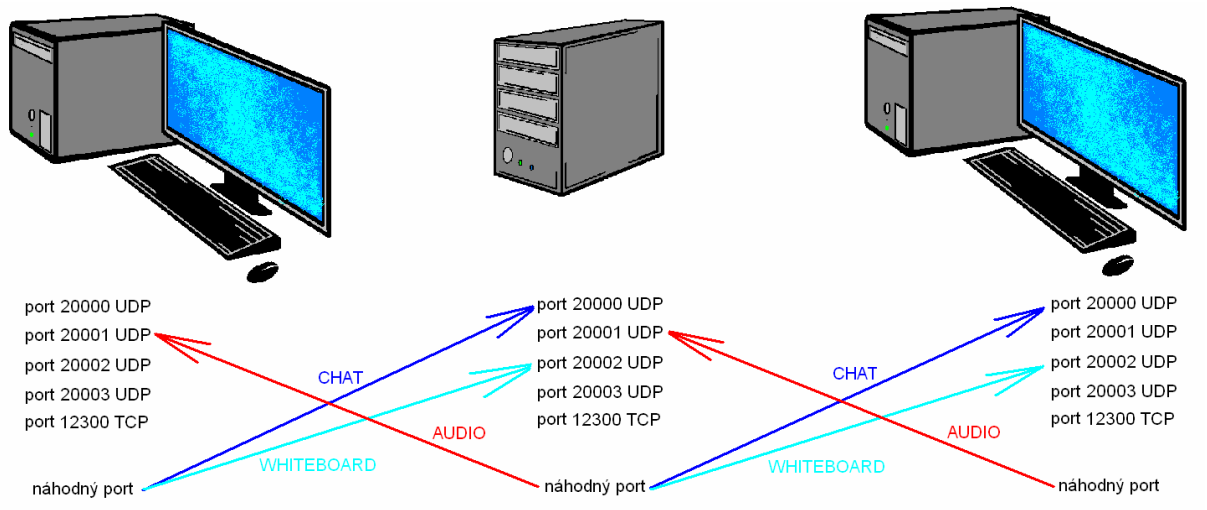
9.2 Vlákna a porty

Serverová aplikace je tedy samozřejmě vícevláknová. Každému nástroji virtuální třídy je přiřazen jeden port a jedno vlákno, které tuto komunikaci obsluhuje. Stručně a výstižně je to popsáno v tabulce 9.1.

Tab. 9.1: Seznam využitých portů.

Využité porty		
Port	Typ	Funkce
20000	UDP	Přenos textových zpráv pro chat.
20001	UDP	Přenos audio paketů.
20002	UDP	Přenos paketů nástroje whiteboard.
20003	UDP	Řídící komunikace (přihlašovaní, obnova seznamu atp.)
12300	TCP	Přenos screenshotů.

Jak lze vidět, tak systém využívá čtyři UDP porty a jeden port TCP (přenos screenshotů přes TCP popsán v kapitole Whiteboard). Pro odesílání paketů se použije vždy nějaký náhodný právě volný port přidělený systémem (obr. 9.2).



Obr. 9.2: Ukázka využití portů..

10 Testování

Každý nový vytvářený software je zapotřebí testovat. Ať už během programování (při „pavučinovém“ typu vývoje) nebo až po dokončení vývoje (při kaskádním typu vývoje). Stejně tak jsem já přistoupil k testování mého systému. Během vývoje jsem si vytvářel verze a při každé nově implementované funkci jsem provedl její testy a občas také regresní testy, zda nebyla novým vývojem narušena nějaká již dříve implementovaná funkce aplikace. Testování takového typu programu sebou nese bohužel nutnost několika počítačů. Dokud byl vývoj ve fázi dvou navzájem komunikujících klientských aplikací, tak stačily dva propojené počítače (z důvodu využívání portů nejde spustit aplikace duplicitně na jednom počítači a pomocí localhost se zas nedaly otestovat některé funkce). Ale s přechodem na více uživatelů a tedy s tím spojenou komunikaci přes server už byly zapotřebí počítače nejméně tři. K průběžnému testování jsem tedy používal jeden desktopový počítač připojený k Internetu na viditelné IP adrese jako server, jeden laptop připojený k němu na lokální síti přes wi-fi a jeden počítač na vzdálené viditelné IP adrese, ke kterému jsem přistupoval pomocí vzdálené plochy. Tedy testování bylo průběžně prováděno nejen v LAN sítí, ale i s klientem v jiném segmentu a také i přes wi-fi. Aplikaci se všemi svými funkcemi jsme otestovali také s vedoucím mé práce Ing. Václavem Pfeiferem při spojení mezi Zlínem a Brnem.

Jak jsem zmínil v kapitole *Náhled do UDP a TCP komunikace*, tak má aplikace není schopná překonat NAT (Network Address Tranlsator), tedy z toho plyne požadavek, aby přihlašující se klient byl na veřejně viditelné IP adrese. Bohužel spousta koncových uživatelů Internetu už je připojena za nějakým domácím routerem a tedy i za NATem a tak nebylo jednoduché zajistit větší počet lidí, kteří by byli v přesnou danou dobu připojení na veřejně viditelné IP adrese a zúčastnili se testování. Provedl jsem tedy testování aplikace na lokální síti ve firemních kancelářích, kdy bylo ve stejnou dobu připojeno deset spolu komunikujících uživatelů a test proběhl bez problémů. Systém fungoval bez komplikací a bez známky nějakého nechtěného zpoždění. Upload ze serveru se pohyboval kolem 600Kb/s. Na fyzickém serveru, přes který jsme komunikovali, byl v dobu testu spuštěný tedy také MySQL server a FTP server (ze kterého jsme v rámci testu s použitím klientské aplikace virtuální třídy také zkoušeli stahovat data a uploadovat data), což z něj ve chvíli testování dělá svým způsobem fyzicky server víceúčelový.

Ve finální verzi tohoto softwaru je sice audio komunikace na méně kvalitní úrovni a asi s půlvteřinovým zpožděním, avšak s bezproblémovou srozumitelností a ostatní nástroje a funkce pracují zcela správně a nebyla nalezena nějaká závažnější chyba.

11 Závěr

V mé práci jsem v první, tedy teoretické části popsal dnešní systémy virtuální třídy a jejich vlastnosti se zaměřením na možnosti jejich softwarového vývoje. Zaměřil jsem se také na popis dnešních možností implementace videokonference v rámci systému virtuální třídy. V druhé, tedy praktické části jsem dle zadání navrhl vlastní systém virtuální třídy, který jsem realizoval pomocí jazyka Java SE.

Výstupem mé práce je tedy funkční systém virtuální třídy, jakožto desktopová klientská Java aplikace komunikující přes centrální serverovou aplikaci téže napsanou v jazyku Java s přídavným využitím relační databáze typu MySQL a FTP serveru. Do systému jsem implementoval komunikační nástroje a funkce, které odpovídají základním požadavkům na běžnou funkční virtuální třídu. Provedl jsem také podrobné testování této třídy spolu s více připojenými uživateli a tedy i se sledováním síťových nároků na provoz tohoto systému.

Seznam použité literatury

[1] FILIP, O. *Úvod do IP multicastu [online]* 2004. Dostupné z: (http://www.lupa.cz/clanky/uvod-do-ip-multicastu/).

[2] BIGELOW, S., J. *Mistrovství v počítačových sítích*. Nakladatelství CPRESS 2004. ISBN: 80-251-0178-9.

[3] BLOCH, J. Java *efektivně – 57 zásad softwarového experta*. Vyd. Grada 2002. ISBN: 80-247-0416-1.

[4] BOMERS, F. ; PFISTERES, M. The *Java Sound Internet Phone* [online] 2004. Dostupné z: (http://www.jsresources.org/apps/3196_InternetPhone.pdf).

[5] DEERING, S. *RFC 1112: Host extensions for IP multicast* [online] 1989. Dostupné z: (http://www.freesoft.org/CIE/RFC/1112/index.htm).

[6] GESTO COMMUNICATIONS *Co je videokonference, proč se vyplatí, jak funguje* [online] 2009. Dospupné z: (http://www.gestocomm.cz/gesto-communications-obor-co-je-videokonference-proc-se-vyplati-jak-funguje.htm).

[7] Global Knowledge Training LLC: *Virtual Classroom e-Learning Technical Support* [online] 2009. Dostupné z: (http://www.globalknowledge.com/training/generic.asp?pageid=133).

[8] HEROUT, P. *Java – bohatství knihoven*. Nakladatelství Kopp 2006. ISBN: 80-7232-288-5.

[9] HEROUT, P. *Java – grafické uživatelské prostředí a čeština*. Nakladatelství Kopp 2006. ISBN: 80-7232-237-0.

[10] HEROUT, P. *Učebnice jazyka Java*. Nakladatelství Kopp 2005. ISBN: 80-7232-115-3.

[11] MAIXNER, D. *Java servlety, JSP a webové servery s jejich podporou, Bakalářská práce* [online]. Dostupné z: (http://www.slunecnice.cz/sw/java-servlety/stahnout/).

[12] MAJER, M. Síťování v Javě: UDP [online] 2006. Dostupné z: (http://www.root.cz/clanky/sitovani-v-jave-udp/).

[13] MALÝ, M. Webové aplikace (a ty další) [online] 2007. Dostupné z: (http://www.misantrop.info/747251-webove-aplikace-a-ty-dalsi.php).

[14] PRAVDA, V.; BAREŠOVÁ A. Virtuální třída jako trend v distančním vzdělávání [online] 2004. Dostupné z: (www.csvs.cz/publikace/NCDiV2004_sbornik/PravdaBaresova-255-258.doc).

[15] PULLEN, J. M. Synchronous *Internet Distributed Education Technology* [online] 2002. Dostupné z: (http://netlab.gmu.edu/NEW/SyncDistEd.pdf).

[16] Sauron Software: *ftp4j 1.3 manual* [online] 2009. Dostupné z: (http://www.sauronsoftware.it/projects/ftp4j/manual.php).

[17] SKŘIVAN, J. *Databáze nejsou jen MySQL* [online]. Dostupné z: (http://interval.cz/clanky/databaze-nejsou-jen-mysql/).

[18] SPELL, B. *JAVA Programujeme profesionálně.* Nakladatelství CPRESS 2002. ISBN: 80-7226-667-5.

[19] Sun Microsystems, Inc. *Java 2 Platform Standard Edition 5.0 API Specification* [online] 2004. Dostupné z: (http://java.sun.com/j2se/1.5.0/docs/api/).

[20] VICHA, K. *Virtuální Studium – iluze či nutnost* [online] 2003. Dostupné z: (http://interval.cz/clanky/virtualni-studium-iluze-ci-nutnost/).

[21] WIKI dokumentace, *Codecs* [online] 2009. Dostupné z: (http://www.voip-info.org/wiki/view/codecs).

Abecední seznam použitých zkratek

API	application programming interface
AWT	abstact window toolkit
BLOB	binary large object
FTP	file tranfer protocol
GUI	graphical user interface
IGMP	internet group management protocol
IP	internet protocol
Java 2 SE	Java 2 standard edition
JMF	java media framework
JSP	java server pages
LAN	local area network
MCU	multipoint control unit
NAT	network address translator
PCM	pulse-code modulation
PHP	personal (professional) homepage / hypertext preprocesor
QoS	quality of service
RTP	real-time protocol
SQL	structured query language
TCP	transmission control protocol
TCP/IP	transmission control protocol / internet protocol
TTL	time to live
UDP	user datagram protocol
VoIP	voice over internet protocol
VT	virtuální třída

Seznam příloh

A – Třídy klientské aplikace

B – Třídy serverové aplikace